

# Módulo 1

## Bases de datos

```
function updatePhotoDescription() {
  if (descriptions.length > (page * 9) + (currentImage - 1)) {
    document.getElementById('bigImageDesc').innerHTML = descriptions[page * 9 + currentImage - 1];
  }
}

function updateAllImages() {
  var i = 1;
  while (i < 10) {
    var elementId = 'foto' + i;
    var elementIdBig = 'bigImage' + i;
    if (page * 9 + i - 1 < photos.length) {
      document.getElementById(elementId).src = 'images/' + photos[page * 9 + i - 1];
      document.getElementById(elementIdBig).src = 'images/big/' + photos[page * 9 + i - 1];
    } else {
      document.getElementById(elementId).src = 'images/' + photos[photos.length - 1];
      document.getElementById(elementIdBig).src = 'images/big/' + photos[photos.length - 1];
    }
    i++;
  }
}
```

# Contenido

<b>UF1: Introducción a las Bases de Datos.....</b>	<b>4</b>
<b>1. Introducción a las Bases de Datos (BBDD) .....</b>	<b>4</b>
Evolución histórica de las BBDD .....	5
Ventajas e inconvenientes de las BBDD.....	6
Almacenamiento de la información .....	6
Gestor de Sistemas de las BBDD.....	10
Arquitectura ANSI/X3/SPARC. Estándar y niveles.....	16
Modelos de BBDD. Tipos: jerárquico, red y relacional .....	17
Bases de datos centralizadas y distribuidas .....	21
<b>2. Modelo Entidad-Relación .....</b>	<b>27</b>
Concepto de modelo entidad-relación .....	28
Entidad: representación gráfica, atributos y tipos de claves .....	28
Relación: representación gráfica, atributos, grado y cardinalidad.....	30
Diagrama Entidad-Relación.....	30
Modelo Entidad-Relación extendido .....	36
Guía para la construcción de un modelo entidad-relación .....	41
<b>3. Modelo Relacional.....</b>	<b>42</b>
Terminología del modelo relacional.....	42
Concepto de relación. Propiedades y relaciones.....	42
Atributos y dominio de los atributos.....	43
Concepto y tipos de clave: candidatas, primarias, ajenas, alternativas .....	45
Otros conceptos: tupla, grado, cardinalidad, valores nulos, comparación con ficheros.....	45
Reglas de integridad: de entidad y referencial.....	46
Traducción del modelo entidad-relación al modelo relacional.....	48
<b>4. Normalización .....</b>	<b>50</b>

<b>Concepto de normalización, dependencias funcionales y sus tipos .....</b>	<b>50</b>
<b>Primera forma normal (1FN) .....</b>	<b>52</b>
<b>Segunda forma normal (2FN) .....</b>	<b>53</b>
<b>Tercera forma normal (3FN).....</b>	<b>54</b>
<b>Forma normal Boycce-Codd .....</b>	<b>55</b>
<b>Otras formas normales (4FN, 5FN) .....</b>	<b>55</b>
<b>Desnormalización .....</b>	<b>55</b>
 <b>UF2: Lenguajes SQL: DML y DDL.....</b>	<b>56</b>
<b>1. Lenguajes de las BBDD. SQL.....</b>	<b>56</b>
<b>1.1 Herramientas para gestionar los datos en un SGBDR corporativo.....</b>	<b>57</b>
<b>1.2 Lenguaje de definición de datos (DDL) .....</b>	<b>65</b>
<b>1.3 Lenguaje de manipulación de datos (DML) .....</b>	<b>72</b>
<b>1.4 Extensiones y otras cláusulas del lenguaje .....</b>	<b>77</b>
<b>1.5 Herramientas de la BDD para optimizar consultas.....</b>	<b>81</b>
<b>2. Estrategias para el control de las transacciones y de la concurrencia.....</b>	<b>82</b>
<b>2.1 Concepto de integridad .....</b>	<b>82</b>
<b>2.2 Concepto de transacción. Control .....</b>	<b>83</b>
<b>2.3 Propiedades de las transacciones: atomicidad, consistencia, aislamiento y permanencia .....</b>	<b>83</b>
<b>2.4 Estados de una transacción: activa, parcialmente comprometida, fallida, abortada y comprometida .....</b>	<b>84</b>
<b>2.5 Problemas derivados de la ejecución concurrente de transacciones.....</b>	<b>84</b>
<b>2.6 Control de concurrencia: técnicas optimistas y pesimistas .....</b>	<b>85</b>
<b>2.7 Recuperación ante errores. Mecanismos para deshacer transacciones.....</b>	<b>87</b>
<b>3. Lenguajes de las BBDD para la creación de su estructura .....</b>	<b>88</b>
<b>3.1 Vistas y otras extensiones del lenguaje.....</b>	<b>88</b>

## UF1: Introducción a las Bases de Datos

### 1. Introducción a las Bases de Datos (BBDD)

Podemos definir las Bases de Datos como una **agrupación de datos que son relevantes para una determinada entidad**.

Entre sus propiedades más importantes podemos destacar las siguientes:

- Representan algún aspecto del mundo real.
- Son diseñadas y almacenan datos para un objetivo específico y están destinada a un grupo de usuarios.



## 1.1. Evolución histórica de las BBDD

Veamos la evolución que han tenido las bases de datos desde sus comienzos hasta las existentes hoy en día:

- **Años sesenta y setenta: Sistemas Centralizados**

Nos encontramos con sistemas diseñados para facilitar el uso de conjuntos de datos muy amplios interrelacionados de forma compleja. Estos sistemas solo trabajaban por lotes (**batch**).

Estos sistemas de gestión de base de datos (SGBD) se utilizaban para grandes empresas como la industria del automóvil y la construcción de naves espaciales.

- **Años ochenta: Sistemas de Gestión de Bases de Datos (SGBD) relacionales**

Fue a partir de los años 80 cuando empezó a extenderse la informática y, con ella, las BBDD. Las aplicaciones que gestionaban las BBDD anteriormente estaban diseñadas para ser tratadas por personal muy cualificado. Para solventar este problema apareció la **estandarización** con un nuevo lenguaje de programación denominado SQL. Este lenguaje produjo un estímulo en el ámbito de las bases de datos relacionales.

Fue entonces cuando aparecieron los primeros ordenadores personales, capaces de manejar sistemas de gestión de BBDD a nivel de usuario.

- **Años noventa: Distribución, C/S y 4GL**

A principios de los noventa nos encontramos con una distribución bastante amplia de utilización en todas las empresas, por lo que esta década se caracteriza por conseguir mejorar el rendimiento de los sistemas gestores.

La idea de SGBD de la época consistía en poder gestionar las distintas BBDD que utiliza cada usuario en un ordenador personal.

- **Tendencias actuales**

Debido a los límites que presentaban las BBDD en épocas anteriores, nace la necesidad de incluir tecnologías con imagen y sonido a nuestro almacén de información.

Por esta razón, los SGBD incluyeron el **Tipo Abstracto de Datos** (TAD). Este tipo de dato no solamente comprende valores, sino también sus operaciones y propiedades.

Esta nueva forma de entender las BBDD nos lleva a la implementación de la **Orientación a Objetos** (OO). El éxito del paradigma de la **Programación Orientada a Objetos (POO)**, la inclusión de las BBDD en las páginas web, hacen que los SGBD relacionales estén en plena transformación para adaptarse a estas nuevas tecnologías de reciente éxito y fuertemente relacionadas.

## 1.2. Ventajas e inconvenientes de las BBDD

- **Ventajas**

- La información contenida en la BDD se procesa de forma independiente. Gracias a un buen diseño de relación entre los datos conseguimos la menor redundancia posible.
- Mediante las BBDD podemos llegar a conseguir la unidad máxima de información partiendo de una pequeña cantidad de datos.
- Aseguran la integridad de los datos y devuelven unos resultados coherentes.
- Ofrecen seguridad para la información.
- Reducen espacio para el almacenamiento de los datos respecto a otros medios (como el papel físico).

- **Desventajas**

- Tienen un mantenimiento costoso debido a que el administrador encargado de la BDD tiene que ser una persona cualificada para tal fin.
- Ofrecen poca rentabilidad a corto plazo debido a la inversión inicial en equipos y personal.
- El tamaño de la BDD requiere bastante espacio en el disco duro y en la memoria principal para gestionarla.

## 1.3. Almacenamiento de la información

Debido a la gran cantidad de demanda de información que necesitamos almacenar y registrar es conveniente el uso de una herramienta capaz de gestionar esos datos de una manera eficiente, fácil y segura.

A continuación, vamos a ver las distintas formas de almacenar y gestionar estos datos:

- **Ficheros planos, indexados, acceso directo.**

Estructuras de información creadas por el sistema operativo (SO) para almacenar datos.

Estos datos suelen estar almacenados de forma permanente en discos duros u otras unidades de disco. Estos ficheros, también llamados archivos, se caracterizan por tener un nombre y una extensión para determinar el tipo de información que contiene.

Por ejemplo, “datos.jpg” se refiere a un fichero que tiene por nombre “datos” y su extensión “.jpg” nos indica que su información es la de una imagen.

El contenido de un fichero puede ser de dos tipos:

- **Texto:** también suelen llamarse ficheros planos. Su información puede ser traducida por el SO a caracteres alfanuméricos y números de acuerdo con la tabla de código ASCII. Así, estos datos son legibles por el usuario.
- **Binario:** la información contenida en ellos suele ser una estructura de datos más compleja que los anteriores ya que almacenan diferente tipo de información. Las imágenes, los vídeos, los archivos comprimidos, los archivos ejecutables, entre otros, son un buen ejemplo. Estos requieren de un formato para poder ser interpretados. Este tipo de datos son los más habituales a la hora de componer una BBDD ya que la información contenida en ellos debe mantener una estructura lógica y organizada para que las aplicaciones destinadas a tal fin puedan acceder sin ningún tipo de problema. Esta estructura sería muy difícil de desarrollar con un fichero de texto.

Las **operaciones básicas** que se pueden realizar en un fichero son:

<b>Abrir (open)</b>	Prepara el fichero para su posterior operación de lectura y/o escritura.
<b>Cerrar (close)</b>	Cierra el fichero por lo que no se puede trabajar más con él.
<b>Leer (read)</b>	Obtiene la información del fichero.
<b>Escribir (write)</b>	Guarda información en el fichero.
<b>Posicionarse (seek)</b>	Posiciona el puntero para su posterior operación de lectura y/o escritura.
<b>Fin de fichero (eof)</b>	Marca el final del fichero.

- **Bases de datos. Conceptos, usos y tipos según modelo de datos y ubicación.**

Podemos definir una **base de datos (BBDD)** como una colección de información perteneciente a un mismo contexto de forma relacionada y organizada.

Se utiliza para mantener la información de diversos objetos de una forma coherente y jerarquizada, de tal manera que, con la mínima cantidad de datos almacenados podamos sacar el mayor número de resultados posible. Para tal fin, vamos a organizar la información dentro de la BBDD en unas estructuras llamadas tablas (definidas posteriormente), que deben estar perfectamente relacionadas entre ellas.

A continuación, se definen de forma más detallada algunos de los conceptos usados en bases de datos:

- **Dato:** cuando hablamos de dato nos referimos a una información concreta sobre algo en específico. Por ejemplo, la edad de una persona, la cantidad de artículos disponibles en un almacén, etc.
- **Tipo de dato:** nos sirve para agrupar los datos según la información que contenga: numérica, alfanumérica, fecha u otros.
- **Campo:** es un identificador que agrupa los datos referidos a una misma información. Por ejemplo: el campo “edad”, englobaría tipos de datos numéricos, cuyos datos podrían ser: 12, 26, 52, etc.
- **Tabla:** corresponde a un elemento, objeto o entidad la cual está formada por varios campos. Por ejemplo, podríamos crear la tabla Persona, que estuviera formada por los campos nombre, apellidos, edad, sexo, DNI, etc.
- **Registro:** un registro es una agrupación de los datos que nos proporcionan toda la información de una tabla. Por ejemplo, en la tabla Persona, un registro sería: María, Gómez, 25, M, 47585858X.
- **Campo Clave:** es un campo que tiene como particularidad ser el identificador de cada registro. Por ejemplo, en la tabla Persona el campo clave podría ser el DNI, ya que este identifica de forma única a cada persona.
- **Consulta:** una consulta es una petición de búsqueda que se realiza cuando se quiere obtener alguna información. También se conoce como *query* y se diferencian consultas de peticiones, inserciones de datos, actualización o eliminación. Estas consultas pueden ser ejecutadas sobre una o varias tablas.
- **Índice:** es una estructura que agrupa los campos clave de cada tabla. De esta forma si buscamos a través de este índice podremos encontrar los registros de una forma más rápida.
- **Vista:** es una tabla virtual que engloba campos de diferentes tablas.
- **Informe:** es un listado de campos y registros seleccionados de forma que facilite su lectura. Suelen pedirse por parte del cliente, por ejemplo, para realizar informes mensuales de datos.
- **Guiones o Scripts:** es un conjunto de instrucciones que se realiza con una cierta funcionalidad. Estas instrucciones suelen realizarse para el mantenimiento de los datos de las tablas.
- **Procedimiento:** un procedimiento es un tipo especial de script que está almacenado en la propia base de datos.

Una vez entendidos estos conceptos, se puede ver a continuación diferentes ejemplos significativos donde se utilizan las **BBDD como una estructura para gestionar la información** almacenada:

- **Bibliotecas:** almacén de datos pertenecientes a libros, autores y lectores perfectamente delimitados en sus correspondientes tablas.
- **Censo:** guarda la información demográfica de habitantes, pueblos, ciudades y países.
- **Científicas:** guardan la información recogida en las muestras para su posterior tratamiento.
- **Administrativas:** cualquier empresa necesita mantener la información totalmente ordenada para su buen funcionamiento.

En referencia a la clasificación de las BBDD, **según su ubicación** las podemos clasificar en los siguientes tipos:

- **Varios ficheros:** la información de estas BBDD va a ocupar varios ficheros para su almacenamiento.
- **Sistemas de ficheros:** datos almacenados en fichero.
- **Jerárquicas:** contiene una estructura de listas y árboles
- **En red:** contiene una estructura de árboles y grafos.
- **Una o varias BBDD:** la información está almacenada en una o varias BBDD según la cantidad de datos a guardar.
- **Relacionales:** BBDD organizadas mediante tablas relacionadas.
- **Orientada a Objetos:** objetos complejos con comportamiento.
- **Varias BBDD almacenadas en varios ordenadores:** almacenan tanta cantidad de información que se necesita el espacio de varios ordenadores para guardar toda la información que requiere:
  - **Distribuidas**
  - **Multidimensionales**

## 1.4. Gestor de Sistemas de las BBDD

Un sistema gestor de base de datos (SGBD) es un programa o software que facilita a la persona encargada de la administración de las BBDD el tratamiento de la información que contiene. Puede realizar operaciones como: diseño, consulta y modificación de dichos datos.

A continuación vamos a ver algunos de los más utilizados:



Gestores de Bases de Datos

- **Funciones, componentes y tipos**

Las funciones de los SGBD las podemos clasificar en tres tipos diferentes:

- **Definición:** especifican los tipos de datos, las estructuras y las restricciones que se van a almacenar asegurando la cohesión e integridad de los mismos.
- **Construcción:** proceso de almacenamiento de datos en algún medio controlado por el SGBD.

En esta función, podemos citar la posibilidad que deben tener estos sistemas de gestión para que dicha base de datos se pueda conectar con el exterior utilizando algún medio. El estándar más utilizado para estos casos es el protocolo ODBC que comunica la base de datos con una aplicación externa.

- **Manipulación:** incluye tareas como la manipulación y consulta de los datos almacenados para obtener una información.

Podemos mencionar las herramientas complementarias que ofrecen estos sistemas de gestión para las estadísticas sobre consultas, actualizaciones e incidencias ocurridas en las BBDD. Los SGBD deben estar implementados para que estas operaciones se realicen de forma sencilla y ofrezcan un gran rendimiento.

Según su capacidad y potencia, los podemos clasificar en dos grandes grupos:

- **SGBD ofimáticos:** destinados a manipular BBDD de uso doméstico o pequeñas empresas. Presentan una interfaz intuitiva y sencilla. El administrador de este sistema no tiene que ser un usuario experto.

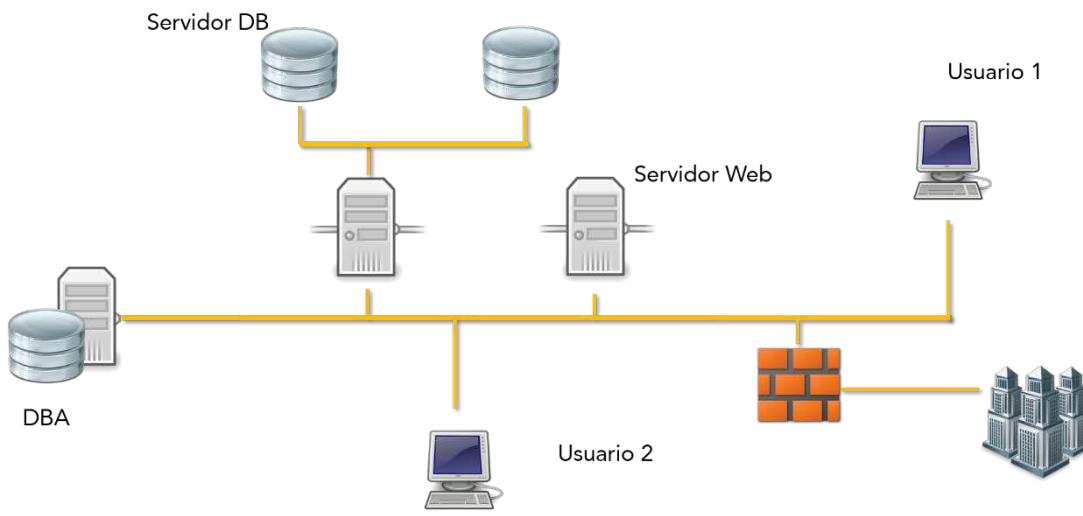
Por ejemplo, **Microsoft Access**.



- **SGBD corporativos:** implementados para manipular BBDD con una capacidad mucho mayor que las anteriores. Suelen ser implantados en grandes o medianas empresas con una gran carga de datos y un volumen alto de transacciones, de tal forma que requieren de un servidor con altas prestaciones.

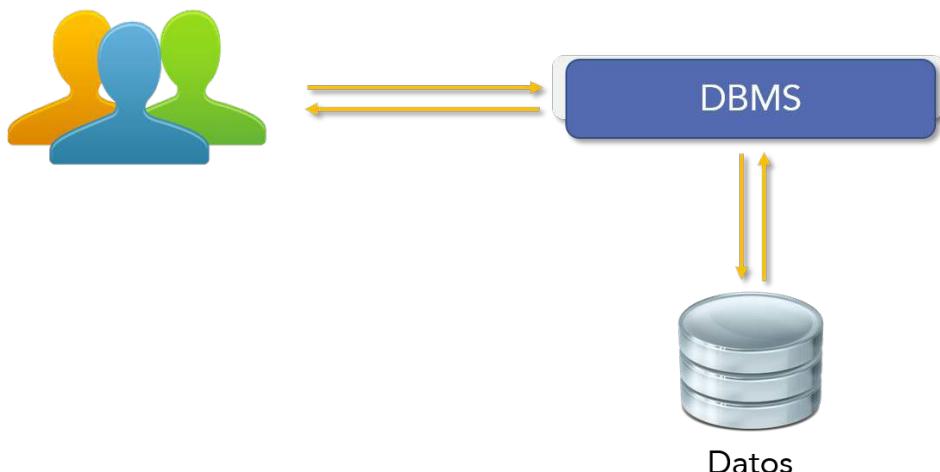
El ejemplo más extendido de estos SGBD es **ORACLE**.

**ORACLE**



- **Objetivos del sistema gestor de la base de datos**

Los sistemas gestores de base de datos (SGBD) *Data Base Management System* son softwares que brindan a los distintos usuarios la posibilidad de procesar, describir, administrar y recuperar aquellos datos almacenados en la BDD. Ofrecen una serie de programas y procedimientos que van a permitir a los usuarios llevar a cabo las diferentes tareas, teniendo siempre especial cuidado con la seguridad de los datos.



*Objetos SGBD*

Para que los SGBD puedan velar por mantener la seguridad e integridad de todos los datos, deben proporcionar una serie de herramientas a los usuarios entre las que podemos encontrar:

- Creación y especificación de los datos: crean la estructura física que se requiera en cada unidad.
- Manipulación de los datos de las BBDD: añaden, modifican, suprimen o consultan los datos.
- Recuperación: se lleva a cabo mediante la creación de copias de seguridad.
- Gestión de la comunicación de la BBDD.
- Creación de aplicaciones.
- Instalación de la BBDD.
- Exportación e importación de datos.

- **Tipos de usuarios de bases de datos: informáticos y no informáticos**

Podemos diferenciarlos de la siguiente manera:

- **Informáticos:** usuarios sofisticados que se comunican con el sistema mediante consultas, incluyendo a los especializados, capaces de diseñar las aplicaciones de la base de datos en diferentes sistemas. Forman parte también de los informáticos los administradores de la base de datos.
- **No informáticos:** aquellos que simplemente interactúan con el programa mediante interfaces de formularios llenando datos.

- **Administrador de la base de datos (DBA): funciones y responsabilidades.**

Nos referimos administradores de la base de datos para referirnos a aquellas personas que tienen el control sobre la base de datos en cuestión. Entre sus funciones principales podemos encontrar:

- Definir el esquema de la BBDD.
- Definir sus estructuras de almacenamiento.
- Modificar dicho esquema y su organización física.
- Asignar autorización para su uso.
- Especificar restricciones de integridad.

- **Tipos de lenguajes de bases de datos.**

Podemos diferenciar cuatro tipos de lenguajes que los detallamos a continuación:

- **DDL (*Data Definition Language*):** es el lenguaje de definición de datos y se utiliza para crear la estructura de una base de datos y, también, para diseñar las vistas del nivel externo. Genera una serie de tablas que se van a almacenar en un archivo al que llamamos diccionario de datos.
- **DML (*Data Manipulation Language*):** es el lenguaje de manipulación de datos y se utiliza para realizar operaciones sobre los datos como insertar, modificar, borrar y consultar. Pueden ser de dos tipos:
  - **Procedimentales:** cuando tenemos que especificar de qué forma se obtienen los datos.
  - **No procedimentales:** cuando solo tenemos que especificar qué datos son los que se van a necesitar.
- **DCL (*Data control Languaje*):** permite crear permisos y roles y así controlar el acceso a la base de datos. Utiliza GRANT para dar privilegios y REVOKE para retirarlos.
- **TCL (*Transactional Control Languaje*):** permite administrar las transacciones que ocurren en la base de datos. Emplea COMMIT para guardar el trabajo realizado y ROLLBACK para deshacer lo realizado desde el último COMMIT.

- **Diccionario de datos: concepto, contenido, tipos y uso.**

Un diccionario de datos es el lugar donde se va a depositar la información referente a los datos que forman la base de datos. Contiene las características lógicas de los sitios donde se van a almacenar los datos del sistema.

El diccionario proporciona información sobre la **estructura lógica y física de la base de datos:**

- **Define todos los objetos** de la base de datos: tablas, vistas, funciones, procedimientos, etc.
- **Define el espacio** que tiene asignado y va a ser utilizado por los objetos.
- **Define los valores** por defecto de las columnas de las tablas.
- **Define los privilegios** asignados.
- Contiene información sobre **restricciones** de integridad.

Aquí podemos ver un ejemplo:

**Nombre:** Empleado      **Fecha de creación:** 27/09/2018

**Descripción:** Información de cada empleado.

Campo	Tipo	Tamaño	Descripción
<b>idEmpleado</b>	Number	20	Campo primario para identificar al cliente
<b>nombre</b>	Varchar	60	Nombre del cliente
<b>apellidos</b>	Varchar	95	Apellido del cliente
<b>salarioE</b>	Float	10	Cantidad salarial anual
<b>departamentoE</b>	Number	5	Número del departamento al que pertenece
<b>fechaEntrada</b>	Date	10	Fecha de incorporación del empleado

**Relaciones:** *departamentoE* con tabla *Departamento*

**Campos Clave:** *idEmpleado*

Además, un diccionario de datos debe cumplir una serie de **características** como:

- Soportar descripciones del modelo conceptual.
- Apoyar la transferencia de información.
- Estar integrado dentro de DBMS.
- Actualizar los cambios en la descripción de la BBDD.
- Estar almacenado mediante un acceso directo para tener una fácil recuperación de información.

Entre los **tipos** de diccionario de datos, podemos encontrar los siguientes:

- **Off-Line:** su función es mantener el diccionario.
- **On-Line:**
  - Funciona con el compilador.
  - No deja que el programador defina los datos, sino que los define directamente.
  - Se asegura de que los datos existen en el diccionario.
  - Añade la definición de los datos.
- **In-Line:** no añade la definición de los datos hasta que no se ejecuta.

## 1.5. Arquitectura ANSI/X3/SPARC. Estándar y niveles

En 1975, el organismo ANSI-SPARC (*American National Standards Institute - Standards Planning And Requirements Committee*) propuso una arquitectura de tres niveles para conseguir separar la vista de los usuarios con el objetivo de ocultar la complejidad de la base de datos. Esta arquitectura oculta los detalles físicos del almacenamiento de los datos a los usuarios.

La mayoría de los sistemas modernos se basan en este sistema, sin embargo, nunca llegó a convertirse en un estándar formal.

Los tres niveles son:

- **Nivel externo o de visión:** es el nivel más próximo a los usuarios. Describe esquemas externos donde se describe la parte de la base de datos que interesa a los usuarios. En esta vista se incluyen los datos relevantes a los que un usuario tiene acceso y también los datos a los que el usuario no tiene acceso.
- **Nivel conceptual:** es el nivel en el que se diseña, mediante un esquema conceptual, la estructura de la BBDD. En este esquema se describen las entidades, atributos, relaciones y operaciones necesarias.
- **Nivel interno o físico:** es el más próximo al almacenamiento de datos en el ordenador. Diseña la estructura física de la base de datos utilizando un esquema interno que detalla cómo se van a almacenar los datos: qué información tienen los archivos, cómo están organizados, métodos de acceso a los registros, longitud y diferentes campos que lo componen.

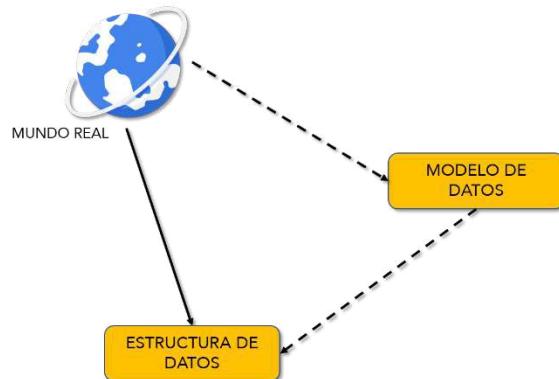


Representación niveles de abstracción de la arquitectura DBMS

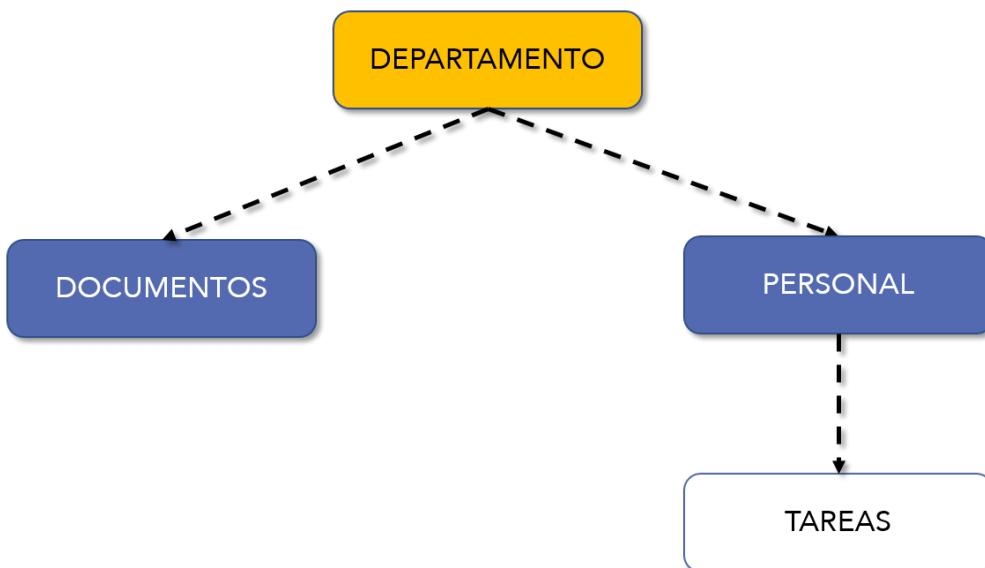
## 1.6. Modelos de BBDD. Tipos: jerárquico, red y relacional

Un modelo de base de datos determina la estructura lógica que tendrá la base de datos, esto quiere decir, establece cómo organizar, almacenar y manipular los datos. Además, también indica qué operaciones se pueden hacer con los datos.

A continuación, se detallan los modelos lógicos más comunes en bases de datos:

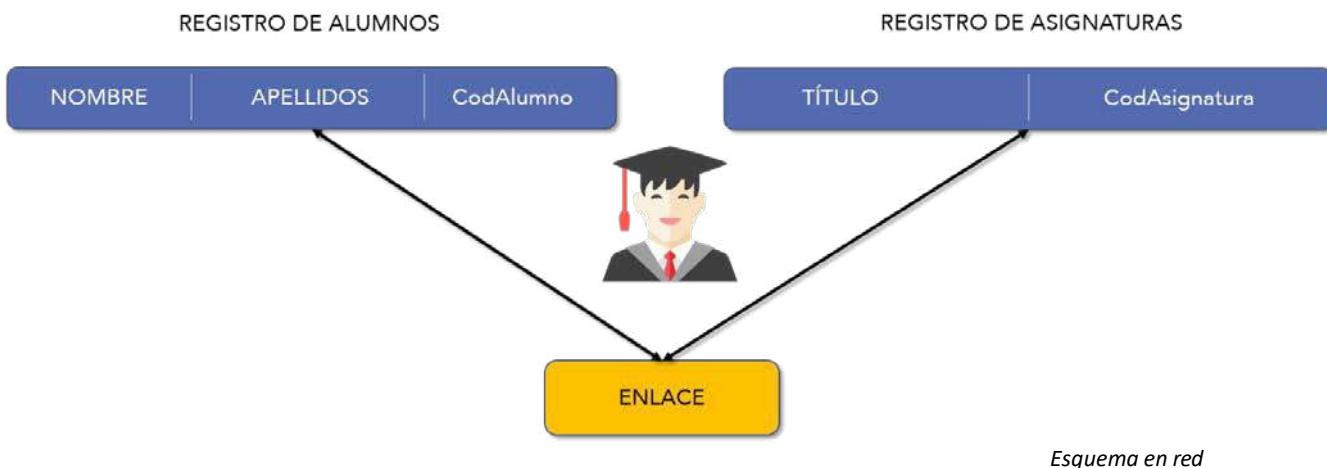


- **Modelo jerárquico:** la información se organiza de manera jerárquica utilizando una relación entre las diferentes entidades siguiendo el tipo padre-hijo. Existe una serie de nodos que contienen atributos que se relacionan con nodos-hijos de manera que puede haber más de un hijo para el mismo padre, pero un hijo no puede tener dos padres.



*Esquema jerárquico*

- **Modelo en red:** este modelo estructura la información en registros (nodos) y enlaces. En los registros se van a almacenar los datos mientras que en los enlaces se podrán relacionar. Este modelo ofrece la posibilidad de tener más de un parente.



- **Modelo relacional:** este modelo se basa en el uso de relaciones. Cada relación representa una unión entre tablas.
- **Modelo entidad-relación:** es el modelo más aceptado hoy en día gracias a las mejoras que ha ido aportando. De todas formas, las distintas variantes del modelo hacen que su representación todavía no sea muy estándar.

La más aceptada actualmente es el modelo **entidad-relación extendido (ERE)** que complementa algunas carencias del modelo original.

No obstante, las diversas variantes del modelo hacen que la representación de este modelo no sea muy común, aunque hay algunas ideas que podemos utilizar en distintas variantes. Hay que insistir en que este modelo no tiene nada que ver con las bases de datos relacionales, los esquemas entidad-relación se pueden utilizar con cualquier SGBD ya que son conceptuales.

- **Modelo orientado a objetos:** este modelo utiliza programación orientada a objetos (POO) ya que ofrece la posibilidad de cohesionar datos y procedimientos. De tal manera que se pueden diseñar estructuras que poseen datos (atributos) en las que se permite definir los diferentes procedimientos (operaciones) que se van a realizar con los atributos.

La programación orientada a objetos (POO) se ideó para bases de datos adaptadas a estos lenguajes. En estos tipos de BBDD se utiliza esta misma idea. A través de este concepto se intenta que estas bases de datos consigan arreglar las limitaciones de las relacionales.

Se supone que son las bases de datos de tercera generación (la primera fue las bases de datos en red y la segunda las relacionales), lo que significa que el futuro parece estar a favor de estas bases de datos.

Aunque siguen sin reemplazar a las relacionales son el tipo de base de datos que más está creciendo en los últimos años.

- **Reglas de integridad de los datos**

Es fundamental que los SGBD puedan garantizar que se va a trabajar de forma segura apostando siempre por la integridad de los datos. Es cierto que la redundancia de estos puede provocar un mal funcionamiento e, incluso, puede que los datos sufran pérdidas de integridad.

Algunos de los motivos principales por los que se puede perder el valor inicial de los datos puede ser por errores provocados por los programas, por los usuarios o por una avería en el sistema, entre otros.

Existen diferentes **reglas de integridad** que se deben tener siempre en cuenta para garantizar un buen funcionamiento:

- **Reglas de integridad del modelo:** deben de cumplirse por el SGBD para que no se pierda integridad cada vez que se realicen actualizaciones de los programas.
- **Reglas de integridad del usuario:** deben de cumplirse por los usuarios ya que en caso de producirse algún error en el que se pierda información, el SGBD está capacitado para dar todas las herramientas necesarias para poderlos reconstruir y, de esta forma, conseguir que no se pierda integridad en el sistema.
- **Modelo distribuido: Ventajas e inconvenientes, técnicas de fragmentación, distribución de datos y esquemas de asignación y replicación de datos.**

Un **modelo de base de datos distribuido (BDD)** constituye un grupo de diferentes máquinas conectadas a través de una red. Cada procesador de los que se utilizan

tiene sus dispositivos de entrada salida y estos intercambian mensajes para conseguir un objetivo común.

Sus **ventajas** principales son:

- **Mejora del rendimiento** ya que permite que varias máquinas conectadas en la red trabajen a la vez con diferentes usuarios.
- Aunque uno de los equipos falle **los demás pueden seguir trabajando**.
- Todos los equipos comparten el **mismo estado**, aunque cada uno trabaje de forma independiente.

Aunque también es importante señalar algunos de los **inconvenientes** que presentan:

- Al estar todos los equipos interconectados en la red, puede que aumente la **pérdida de mensajes y la saturación**.
- **Menor confidencialidad** de los datos.

En un modelo distribuido podemos encontrar diferentes técnicas utilizadas para la **fragmentación**:

- **Horizontal:** se basa fundamentalmente en particionar tuplas en subconjuntos. De esta manera, cada subconjunto debe contener aquellos datos que son comunes. Así, cada fragmento, se puede definir como una operación de selección.
- **Vertical:** la fragmentación vertical se basa en subdividir los atributos en grupos. De esta manera podemos obtener los distintos fragmentos si proyectamos la relación global sobre todos los grupos. Esta fragmentación es exitosa si se puede localizar a cada atributo en, al menos, otro atributo del fragmento en cuestión.
- **Mixta:** en este tipo de fragmentación se van a combinar la horizontal con la vertical o viceversa.

## 1.7. Bases de datos centralizadas y distribuidas

Las **Bases de Datos Centralizadas**, son bases de datos que están almacenadas completamente en un solo lugar, es decir, en una misma máquina. Sin embargo y tal y como se comentaba en el apartado anterior, una **Base de Datos Distribuida (BDD)** es un conjunto de bases de datos que se encuentran lógicamente relacionadas lo que significa que se distribuyen en diferentes sitios de forma que necesita una interconexión de red para comunicarse.

A continuación, detallamos con más precisión cada una de ellas:

- **Centralizadas**

<b>Definición</b>	<ul style="list-style-type: none"> <li>- Las bases de datos centralizadas son aquellas que se encuentran <b>almacenadas en una única computadora</b> por lo que el sistema informático no interacciona con ninguna otra máquina.</li> <li>- Ejemplos de estos sistemas pueden ser bases de datos básicas de un solo usuario o bases de datos de alto rendimiento implantadas en grandes sistemas.</li> </ul>
<b>Características</b>	<ul style="list-style-type: none"> <li>- Almacena todos los componentes en una única máquina.</li> <li>- No tiene demasiados elementos de procesamiento.</li> <li>- Componentes: datos, software de gestión de BDD y dispositivos de almacenamiento.</li> </ul>
<b>Ventajas</b>	<ul style="list-style-type: none"> <li>- No presenta redundancia ni inconsistencia ya que se focaliza todo en un sistema central. Si se tratara de una BDD no centralizada existiría redundancia de la información y, por lo tanto, problemas con el espacio de almacenamiento.</li> <li>- Puede aplicar restricciones de seguridad.</li> <li>- Rendimiento óptimo al procesar datos.</li> <li>- Se evita la inconsistencia de los datos ya que solo existe una sola entrada para cada dato almacenado.</li> </ul>
<b>Inconvenientes</b>	<ul style="list-style-type: none"> <li>- Ante un problema la recuperación de datos es complicada.</li> <li>- No existe la posibilidad de repartir las tareas al intervenir solamente una máquina.</li> <li>- Si un sistema falla perdemos la disponibilidad de la información.</li> <li>- Los <i>mainframes</i> (ordenadores centrales) ofrecen una relación entre el precio y el rendimiento bastante costosas.</li> </ul>

- **Distribuidas**

<b>Definición</b>	<ul style="list-style-type: none"> <li>- Las bases de datos distribuidas son aquellas en las que interviene un conjunto de múltiples BDD relacionadas. Se encuentran en diferentes espacios lógicos y geográficos, pero están interconectadas por una red.</li> <li>- Estas Bases de Datos son capaces de procesar de una forma autónoma, es decir, pueden trabajar de forma local o distribuida.</li> </ul>
<b>Características</b>	<ul style="list-style-type: none"> <li>- Autonomía: los componentes, el SO y la red son independientes y cada uno realiza las diferentes operaciones desde su propio sitio.</li> <li>- No necesita depender de una red central para obtener un servicio.</li> <li>- Presenta la posibilidad de leer y escribir datos ubicados en lugares diferentes de la red.</li> <li>- Puede convertir transacciones de usuarios en instrucciones para manipular datos.</li> </ul>
<b>Ventajas</b>	<ul style="list-style-type: none"> <li>- Acceso rápido.</li> <li>- Al intervenir varios nodos el procesamiento es más rápido.</li> <li>- Los nuevos nodos que intervengan se crean de forma rápida y fácil.</li> <li>- Mejora la comunicación entre distintos nodos.</li> <li>- Refleja una estructura organizativa donde las bases de datos se almacenan en los departamentos donde tienen relación.</li> <li>- Bajo coste a la hora de crear una red de pequeñas computadoras.</li> <li>- Al presentar una red de bases de datos se implementa de forma modular. Esto hace que las operaciones de modificar, insertar o eliminar alguna Bases de Datos sean mucho más fáciles que en el ejemplo anterior.</li> </ul>
<b>Inconvenientes</b>	<ul style="list-style-type: none"> <li>- Presenta una estructura de diseño más compleja.</li> <li>- Aumenta el riesgo de violaciones de seguridad.</li> <li>- Mecanismos de recuperación más complejos debido a que existen muchos más datos.</li> </ul>

- **Componentes: hardware y software**

A nivel de componentes **hardware**, las bases de datos distribuidas suponen un mayor uso de infraestructura ya que, a diferencia de las bases de datos centralizadas, las bases de datos distribuidas ubican sus datos en más de una máquina (denominados nodos o sitios).

En referencia al **software** necesario, las bases de datos distribuidas deben interconectar los nodos que lo componen por lo que necesitan de una red a través de la cual transmitir la información entre los mismos.

- **Niveles de procesamiento de consultas: procesadores locales y distribuidos**

En los **procesadores locales** solamente se hace referencia tanto a tablas como a datos locales, es decir, a aquellos que pertenecen a una misma instancia en una única máquina. Las subconsultas que se ejecutan en un nodo (consulta local) se van a optimizar utilizando el esquema local del nodo. Los diferentes algoritmos se pueden elegir para ejecutar las operaciones relacionales.

Mientras que, en los **procesadores distribuidos**, el objetivo principal va a ser pasar las transacciones de usuario a instrucciones para poder manipular los datos. El principal problema que presentan es de optimización, ya que se determina el orden en el que se realizan el mínimo número de operaciones.

**El procesamiento de consultas es bastante más complicado en procesadores distribuidos que en los locales. Esto se debe a que en las distribuidas interviene un gran número de parámetros que pueden afectar al rendimiento de las consultas que se tengan que realizar.**

La función más importante de un procesador de consultas relacionales va a ser transformar una consulta en una especificación de nivel alto (normalmente en cálculo relacional) en otra equivalente de bajo nivel (normalmente en álgebra relacional).

Esta transformación se debe llevar a cabo y, si se consigue, debe ser perfectamente correcta y eficiente.

- **Bloqueo y concurrencia. Transacciones distribuidas**

Cuando nos referimos a los SGBD debemos señalar que son sistemas concurrentes. Estos sistemas van a ejecutar sus consultas y estas se van a ir procesando al mismo tiempo.

Las transacciones distribuidas se pueden definir como transacciones planas o anidadas que pueden acceder a objetos que han sido administrados por diferentes servidores. Cuando una transacción distribuida finaliza, esta necesita que todos los servidores que han formado parte del proceso verifiquen su buen funcionamiento. En caso de que no se pueda verificar debe abortar la transacción. Este mecanismo se puede llevar a cabo gracias a su propiedad de atomicidad.

- **Distribución de los datos: replicados, particionados, híbridos**

La distribución de los datos es una tarea que corresponde al diseñador. Este se va a encargar de elegir dónde se va a posicionar y qué esquema va a representar dicha base de datos.

Las principales son: replicadas, particionadas e híbridas.

1. **Replicadas:** es un esquema muy costoso ya que cada nodo va a tener información duplicada. Es más lento, porque tiene muchos datos a almacenar, pero merece la pena a largo plazo, ya que va a tener mucha disponibilidad a la hora de leer la información.
2. **Particionadas:** en este caso sólo tenemos una copia de cada nodo. De todas formas, cada nodo alojará algunos fragmentos de la base de datos. Esto hace que el coste sea más reducido, aunque, también, va a tener menos disponibilidad que el anterior.
3. **Híbridas:** aquí vamos a representar la partición y replicación del sistema.

- **Seguridad y recuperación de la información en bases de datos distribuidas**

Existen diferentes tipos de ataques a la seguridad entre los que podemos destacar: la privacidad y confidencialidad de los datos, los que están asociados a la autenticación y los que deniegan al servicio.

En cuanto a las herramientas de seguridad, debemos señalar los distintos protocolos de seguridad, el cifrado de las claves y los cortafuegos.

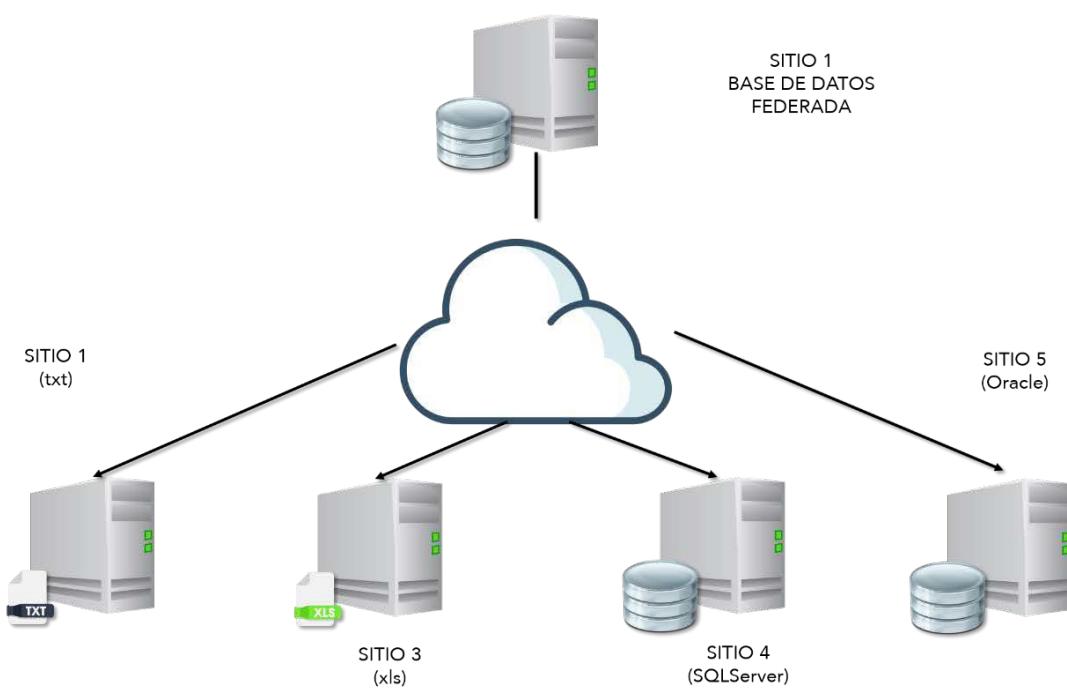
Para poder recuperar los datos debemos tener activa la tolerancia a fallos que permite que, en caso de fallo de algún componente, el sistema siga funcionando de forma correcta.

- **Arquitectura-implementaciones: múltiples y federadas**

Las **bases de datos federadas** son un conjunto de sistemas de bases de datos que trabajan de forma cooperativa y autónoma.

Los usuarios, gracias a una interfaz, pueden acceder a los datos. Esta interfaz no tiene diseñado un esquema total donde estén todos los datos, sino que contiene diferentes esquemas más pequeños que hay que unificar.

Las **BBDD federadas** parecen BBDD normales, pero no tienen existencia física por sí solas, son una vista lógica.



*Arquitectura.*

Las **bases de datos múltiples** actúan como una interfaz de varios componentes diferentes. Una base de datos múltiple cuenta con distintas operaciones que facilitan el acceso a la información, manteniendo la consistencia de ésta y ofreciendo un acceso uniforme a los servicios.

- **Diseño y gestión de BD distribuidas**

Cuando necesitemos diseñar una base de datos distribuidas necesitaremos realizar una serie de pasos que nos permitan tener el diseño correcto para almacenar nuestros datos. Existen una serie de pasos que debemos seguir a la hora de diseñar una base de datos distribuida, veamos los más importantes:

- 1) **Diseñar el esquema conceptual:** Para comenzar necesitaremos detallar el esquema conceptual de toda la base de datos.
- 2) **Diseñar la BD:** Posteriormente, organizar el esquema conceptual establecer sus métodos de acceso.
- 3) **Diseñar fragmentación:** Necesitaremos fragmentar, es decir, realizar subdivisiones en fragmentos de las diferentes partes de la base de datos.
- 4) **Diseñar asignación de fragmentos:** Por último, organizar y seleccionar cómo se van a unir los diferentes fragmentos previamente creados.

Para después gestionar correctamente nuestra base de datos, necesitaremos tener una base de datos estable cuyas relaciones sean coherentes y cuya interconexión entre sus nodos sea correcta. Para administrar las transacciones en bases de datos distribuidas podemos utilizar lo que se conoce como **administrador de transacciones distribuidas (DTM)**. Esto es un programa que procesa y coordina las consultas o transacciones de nuestra base de datos.

## 2. Modelo Entidad-Relación

El modelo entidad-relación se compone de una serie de objetos (entidades) que tienen una relación entre ellos. Este modelo es muy útil para la representación de una base de datos.

Veamos un ejemplo útil de base de datos en el que utilizaremos el modelo entidad-relación:

En unos grandes almacenes se tiene que publicar un folleto de comercial. Gracias a la base de datos de la empresa, el departamento encargado sabrá qué productos consumen sus clientes y en qué fechas lo hacen. Gracias a esto ofertan los productos más consumidos y pueden también consiguen fidelizar a sus clientes y atraer a compradores potenciales.

Para conseguir una base de datos adecuada para el anterior ejemplo, debemos seguir el proceso siguiente:

1. Lo primero de todo es **recabar información del cliente** para saber sus necesidades.
2. Una vez realizadas las entrevistas pertinentes, empezaremos con el **diseño del modelo entidad-relación** (modelo conceptual). En él plasmaremos en forma de entidades, atributos y relaciones todos los requerimientos recogidos en la fase previa.
3. A continuación, transformaremos este diagrama **en el modelo relacional** (basado en relaciones o tablas).
4. Después de crear nuestro segundo diagrama, llega el momento de **decidir qué SGBD debemos de utilizar**.
5. Como último paso, **implementaremos el modelo relacional en nuestro programa** que gestiona la BD (modelo físico).

## 2.1. Concepto de modelo entidad-relación

Como ya introdujimos anteriormente, el modelo entidad-relación (E-R) es un arquetipo conceptual que representa un problema planteado a través de entidades y relaciones.

Ejemplo:



**ENTIDADES:** Cada uno de los objetos de los que almacenamos los datos

**RELACIÓN:** Asociación entre entidades

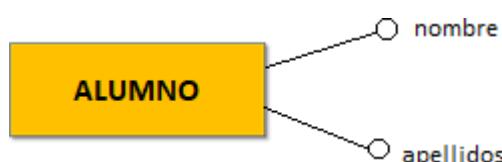
## 2.2. Entidad: representación gráfica, atributos y tipos de claves

Podemos definir las **entidades** como la representación de aquellos elementos (físicos o abstractos) de los que se desea almacenar la información.

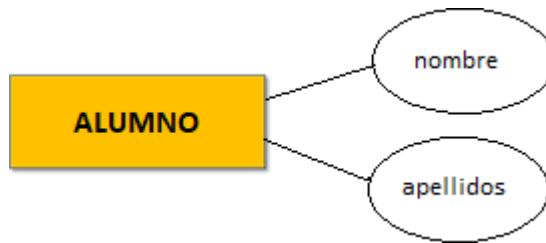
Se pueden representar gráficamente mediante un **rectángulo** que contiene en su interior el nombre del elemento al que representan. Este nombre debe ser único, es decir, no puede aparecer repetido en nuestro diagrama.



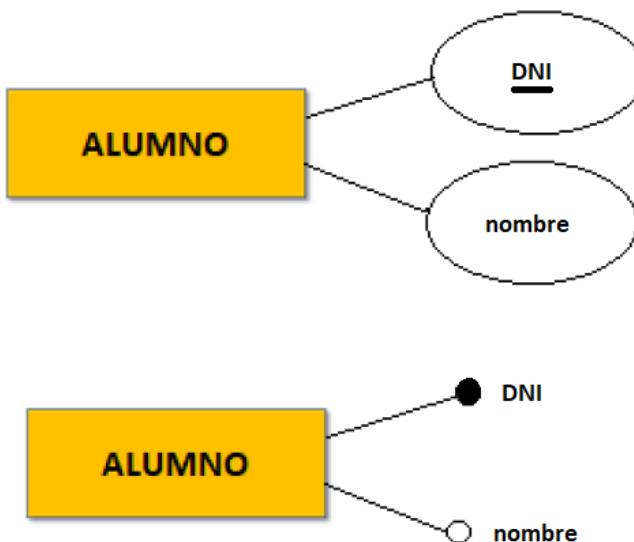
Cuando hablamos de **atributos de una entidad** hacemos referencia a las propiedades o características que tiene una instancia en particular de esa entidad, por ejemplo: título, autor, nombre o fecha. Un ejemplo de un elemento en el que se representan sus atributos sería el siguiente:



O también puede representarse:

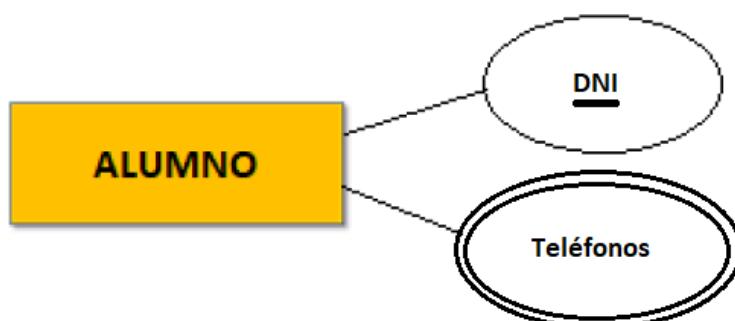


Con relación a los atributos, hemos de saber que un atributo o un conjunto de atributos puede conformar una **clave primaria**, la cual permite identificar de manera única un registro de una tabla. Este concepto lo ampliaremos más adelante, y su representación en el diagrama entidad-relación es la siguiente:



Estas son dos formas de representar la clave primaria en el diagrama entidad-relación. En este caso, la clave primaria de la entidad *Alumno* es *DNI*.

Un atributo puede ser multivalorado para una misma entidad, el atributo posee varios valores. En tal caso, se representa con una doble circunferencia:



### 2.3. Relación: representación gráfica, atributos, grado y cardinalidad

La relación sirve para **escenificar las conexiones entre las diferentes entidades**, dándoles así un significado semántico más completo.

Debe estar identificada con un nombre que indica la relación existente entre las distintas entidades. En la mayoría de los casos se trata de un verbo.

Se pueden representar gráficamente mediante un **rombo**:



### 2.4. Diagrama Entidad-Relación

A continuación, nos adentramos en distintos conceptos, características o funcionalidades que presentan los diagramas entidad-relación.

- **Cardinalidad**

La cardinalidad representa la participación que hay entre las entidades, es decir, el número de entidades con la cual se puede relacionar. Estas son: de uno a uno, de uno a muchos, de muchos a uno o de muchos a muchos.

Se utilizan las siguientes cardinalidades:

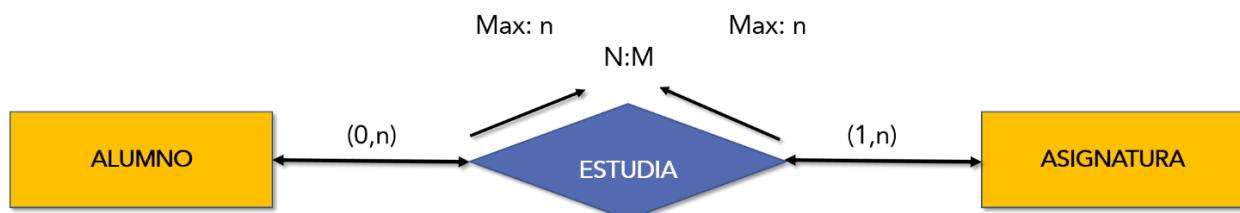
- **0**: si la entidad no está obligada a participar en la relación.
- **1**: si la entidad está obligada a participar y participa de forma obligatoria una vez.
- **N, M, \***: si la entidad no está obligada a participar pero puede hacerlo un número indeterminado de veces.

Veamos en la siguiente tabla la agrupación de valores habitual:

CARDINALIDAD	EXPLICACIÓN
<b>1:1 (Uno a uno)</b>	Una entidad A puede estar vinculada únicamente a una ocurrencia de una entidad B.
<b>1:N (Uno a muchos)</b>	Una ocurrencia de la entidad A puede estar vinculada a varias ocurrencias de la entidad B. Sin embargo, una ocurrencia de la entidad B estará vinculada con una única ocurrencia de A.

<b>N:1 (Muchos a uno)</b>	Una ocurrencia de la entidad A solo puede estar relacionada con una ocurrencia de la entidad B, sin embargo, una ocurrencia de la entidad B puede estar vinculada a varias ocurrencias de la entidad A.
<b>N:M (Muchos a muchos)</b>	Una ocurrencia de la entidad A puede estar relacionada con varias ocurrencias de la entidad B y viceversa.

A continuación un ejemplo:



En el ejemplo, podemos ver que la relación “estudia” viene dada por los máximos de las dos entidades que participan en ella.

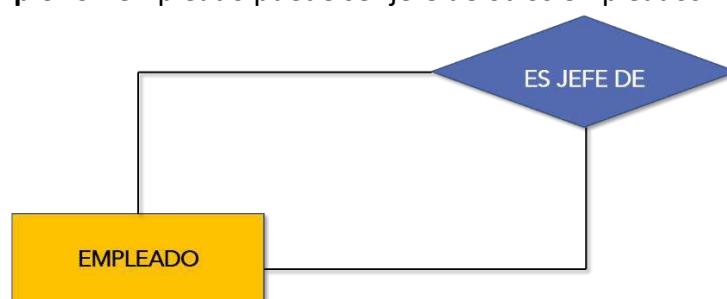
**Nota:** Si hay que poner una segunda N, la segunda N se transforma en una M (se hace así por convenio), pero su significado se mantiene invariable.

- **Tipos de correspondencias en las relaciones: binaria, reflexiva y otros.**

En los diagramas entidad-relación podemos diferenciar entre diferentes tipos de relaciones:

- **Relaciones unarias o reflexivas:** tipo de relación en la que sólo participa una entidad asumiendo diferentes roles dependiendo del sentido de la relación. También llamadas de anillo o grado 1.

**En el ejemplo:** Un empleado puede ser jefe de otros empleados.



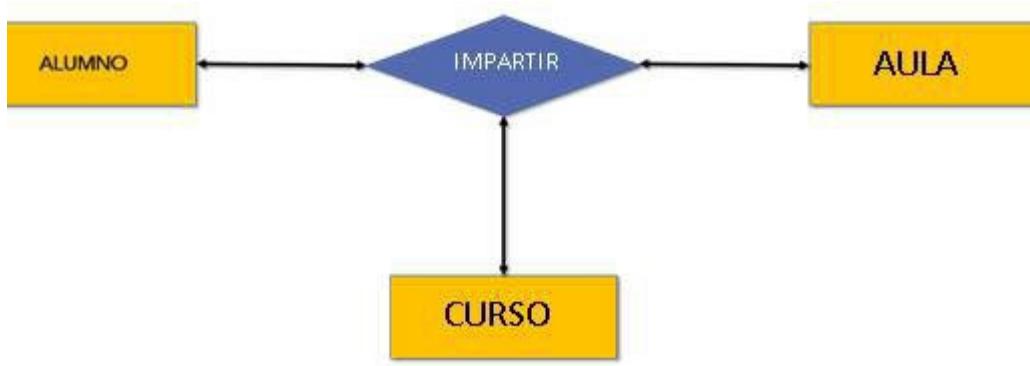
- Relaciones **binarias** o de grado 2: intervienen dos entidades.

**En el ejemplo:** Un alumno estudia una asignatura y una asignatura es estudiada por un alumno.



- Relaciones **ternarias** o de grado 3: intervienen tres entidades.

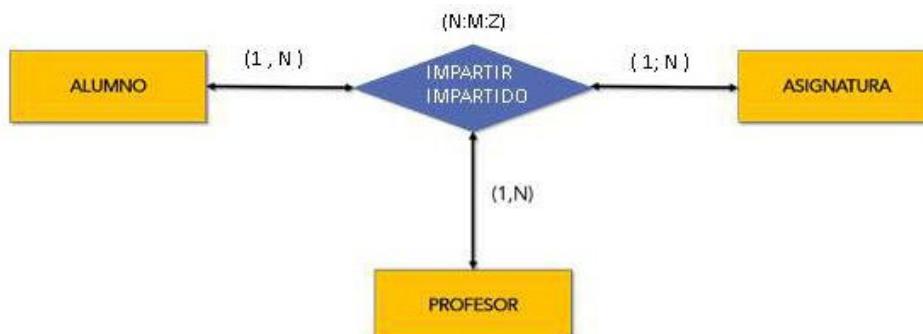
**En el ejemplo:** Un curso puede ser impartido X veces. Un aula puede alojar X veces dichos cursos impartidos. Finalmente, una impartición (asignatura es impartida) de dicho curso en dicha aula puede tener X alumnos.



- Relaciones N-arias:** relaciones en las que intervienen **más de tres entidades**. No suelen utilizarse y suelen descomponerse en relaciones de menor grado.

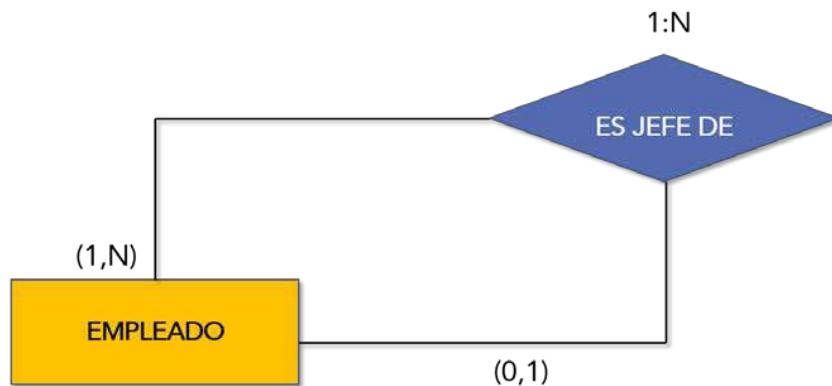
En la cardinalidad de **relaciones binarias** se debe coger el máximo valor de cada participación.

La **cardinalidad en relaciones terciarias y N-arias** se hacen de forma similar a las binarias, debemos escoger el máximo valor de cada participación.



\*\*Establecemos (1,N) en ALUMNO porque entendemos que un PROFESOR debe impartir una ASIGNATURA a al menos 1 ALUMNO, si no, dejaría de tener sentido. También podemos pensar que una asignatura puede no tener alumnos, pero entonces tampoco tendría sentido que existiera la figura de profesor como (1,N), por ello en este ejemplo y supuesto, estimaremos de esta manera las relaciones.

La **cardinalidad en relaciones reflexivas** se lleva a cabo cuando una entidad toma varios roles. Aunque, a la hora de calcular su cardinalidad seguimos tomando como base el mismo principio, es decir, tomamos los máximos de ambas participaciones.



- **Tipos participación de una entidad: obligatoria-opcional (valores nulos)**

Cuando hablamos de participación nos referimos al número máximo de veces que una entidad puede participar en una relación. Los posibles valores que puede adoptar la participación pueden ser:

Participación	Descripción
(0,1)	Mínimo cero, máximo uno.
(1,1)	Obliga a la participación.
(0,n)	Mínimo cero, máximo n (Indefinido)
(1,n)	Mínimo uno, máximo n (Indefinido)

PARTICIPACIÓN	DESCRIPCIÓN
(0,1)	Mínimo cero, máximo uno.
(1,1)	Obliga a la participación.
(0,n)	Mínimo cero, máximo n (indefinido).
(1,n)	Mínimo uno, máximo n (indefinido).

La notación que utilizamos para expresar la participación de un diagrama E/R consiste en colocar la participación al lado de la entidad, sobre la línea de la relación.



Para facilitar su comprensión podemos leerlo como si se tratase de una oración. De este modo, nuestra oración comienza con la entidad sujeto y termina con la entidad que actúe de complemento.

Según el ejemplo, quedaría de la siguiente forma:

- Un “ALUMNO” estudia mínimo una ASIGNATURA y máximo n.
- Una asignatura es estudiada mínimo por 0 alumnos y máximo por n.

- **Entidades fuertes y débiles**

Podemos hacer diferencia entre **dos tipos de entidades**:

1. **Entidad fuerte**: tiene existencia por sí misma, es decir, está dotada de significado propio.

ENTIDAD

2. **Entidad débil**: entidad cuyos atributos no la identifican completamente. Su participación va ligada a una relación fuerte para que esta le ayude a identificarla.

ENTIDAD

*Podemos poner como ejemplo el siguiente caso práctico:*

*Supongamos que una empresa almacena en su BBDD información sobre sus empleados pero que, además, también tiene constancia de los familiares que este tiene. En ese caso contaría con dos entidades: "empleados" y "familiar". La entidad "familiar" sería, en este caso, la entidad débil ya que su existencia depende por completo de la entidad "empleado" (entidad fuerte). En definitiva, sin la entidad "empleado" no existiría la entidad "familiar".*

## 2.5. Modelo Entidad-Relación extendido

Este modelo presentó bastantes limitaciones ya desde sus comienzos debido, sobre todo, a las tecnologías del momento.

Este problema se ha ido solventando con el paso de los años hasta conseguir un nivel adecuado para los diseñadores de las BBDD. Ha incorporado todos los elementos del modelo entidad-relación con todos los conceptos de subclase y superclase, junto a los de especialización y generalización. Todos estos factores han dado lugar al modelo entidad-relación extendido.

Este modelo puede representar bastantes más restricciones en el mundo real, tales como: atributos derivados, generalización/especialización, agregación, exclusividad, exclusión, inclusividad, inclusión.

- **Atributos derivados**

Cuando un atributo está asociado a otro podemos decir que es redundante. Los atributos derivados se obtienen a partir de otros ya existentes. Deben incluirse en la etapa de diseño, aunque si se incluyen en el modelo entidad-relación es únicamente por razones semánticas.

- **Generalización/especialización**

La generalización es un caso especial de relación entre diferentes tipos de entidad que denominaremos subtipos, y otros de carácter más general que vamos a definir como supertipos. La relación que se va a establecer entre ambos es del tipo: "es un" o "es un tipo de" o "*is a*".

Esta jerarquía nos la podemos encontrar de dos formas distintas:

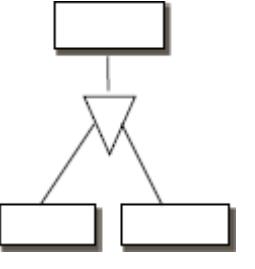
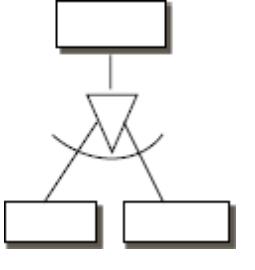
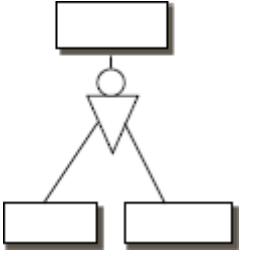
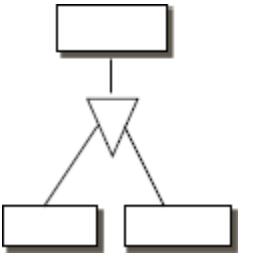
- **Generalización**

Cuando dos o más tipos de entidades van a compartir diferentes atributos. Así podemos deducir que va a existir una entidad supertipo que va a ser la que contenga aquellos atributos comunes a los subtipos.

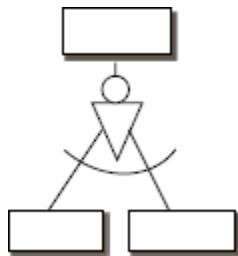
- **Especialización**

Cuando un tipo de entidad tiene algunos atributos que tienen sentido para algunos ejemplares, pero no para todos, es necesario definir subtipos que contengan estos atributos y dejar aquellos que sean comunes para todos en el supertipo.

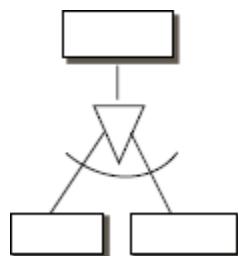
Una vez ya hayamos visto si nos encontramos ante un caso de generalización o de especialización, puede que tengamos las diferentes **restricciones semánticas**:

<b>Solapamiento (inclusiva)</b> Cuando un ejemplar del supertipo puede que pertenezca a más de un subtipo.	
<b>Exclusiva (disjunta)</b> Cuando un ejemplar del supertipo sólo puede pertenecer a un subtipo.	
<b>Totalidad</b> Cuando todo ejemplar del supertipo debe pertenecer a algún subtipo.	
<b>Parcialidad</b> Cuando hay ejemplares del supertipo que no pertenecen a ningún subtipo.	

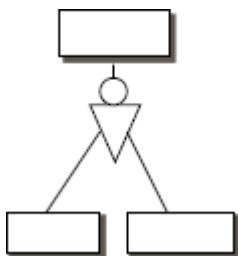
Por tanto, los diferentes tipos en los que podemos diferenciar una relación jerárquica van a ser los siguientes:



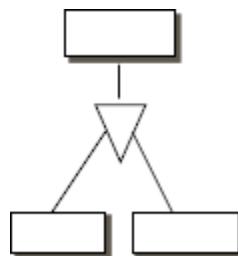
Exclusiva total



Exclusiva parcial



Solapada total

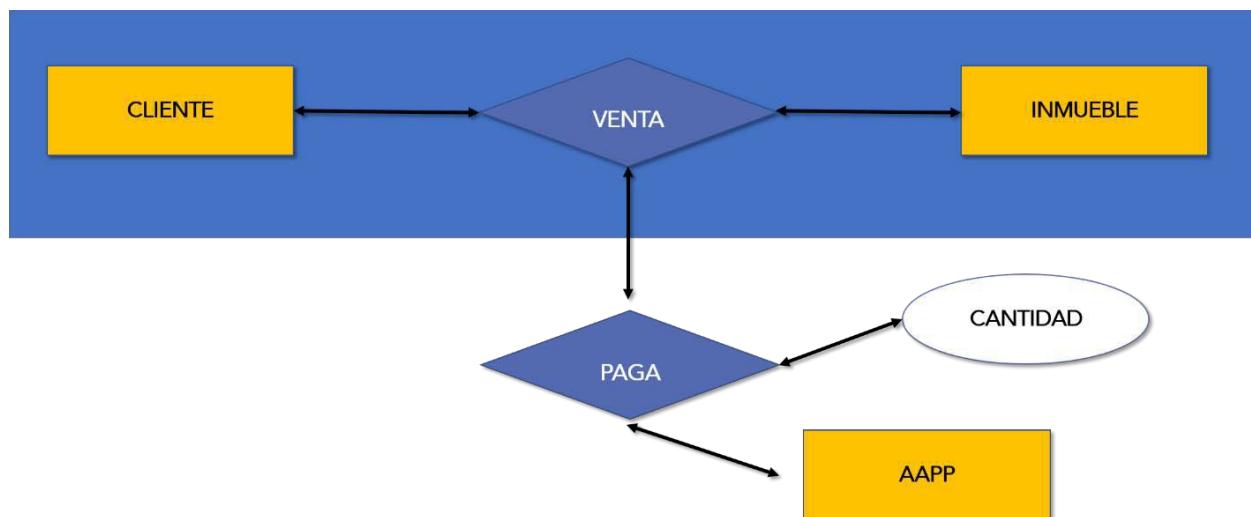


Solapada parcial

- **Agregación**

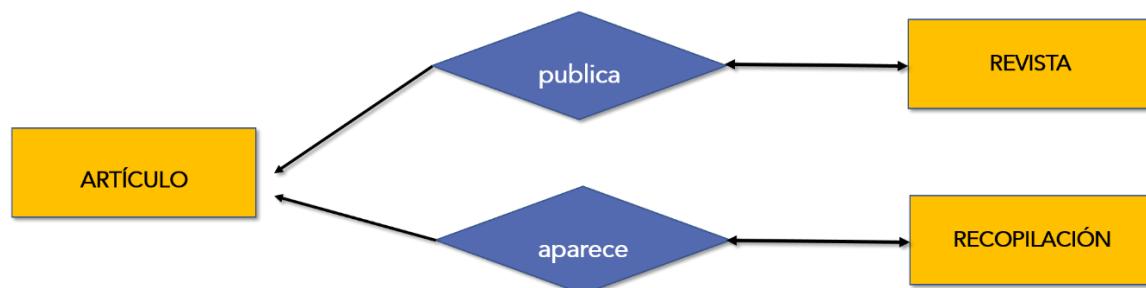
Cuando deseamos que una relación formada por entidades se comporte como entidad para ser relacionada con otras entidades.

*Ej.: Un cliente se compra un piso. Por esa venta a la Administración Pública se la paga una cantidad. Si el cliente no se hubiera comprado ese piso, no se le pagaría dicha cantidad.*



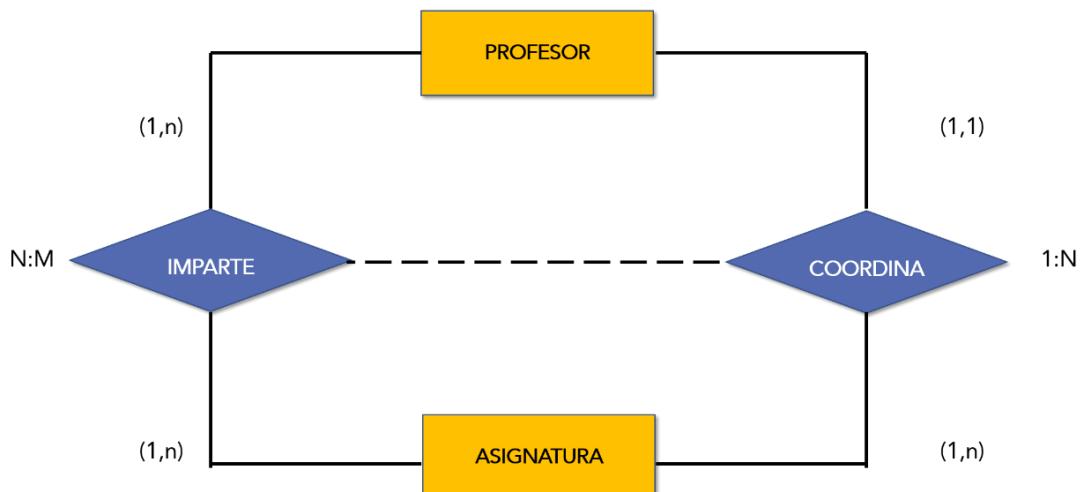
- **Exclusividad**

Se da exclusividad de dos o más tipos de interrelación con respecto a una entidad cuando cada ocurrencia sólo pertenece a uno de los dos tipos de interrelación.



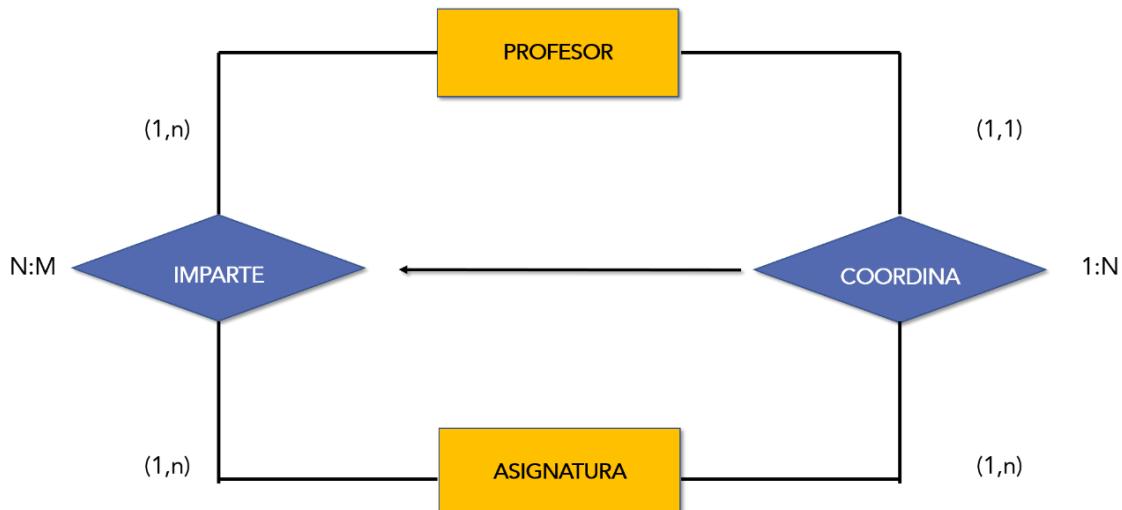
- **Exclusión**

Cuando tenemos una ocurrencia de un determinado tipo de entidad relacionada con otra ocurrencia de otro tipo de entidad diferente mediante una interrelación, no se puede relacionar con esa ocurrencia por ningún otro tipo de interrelación.



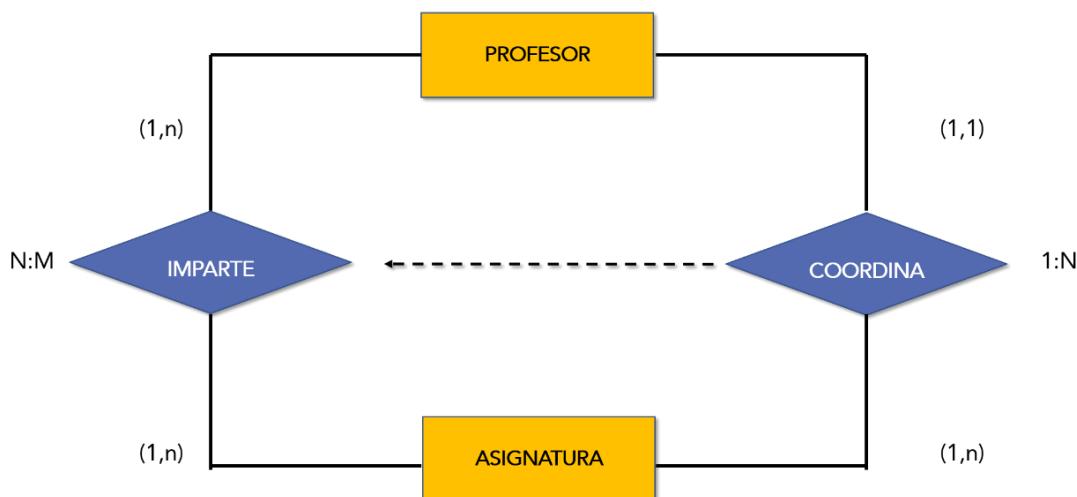
- **Inclusividad**

Para que toda ocurrencia de un tipo de entidad tenga interrelación debe formar parte, necesariamente, de otro tipo de interrelación.



- **Inclusión**

Se da el caso de inclusión cuando una ocurrencia de un tipo de entidad se relaciona con otra ocurrencia de otro tipo de entidad por una interrelación, necesariamente se debe relacionar con esa misma ocurrencia por una interrelación diferente.



## 2.6. Guía para la construcción de un modelo entidad-relación

A la hora de realizar la construcción de un modelo entidad-relación, es importante poner atención para obtener un modelo óptimo y evitar errores. Para ello hacemos a continuación una recopilación de los siguientes pasos:

- 1) Leer el problema las veces que haga falta hasta entenderlo por completo.
- 2) Identificar los diferentes tipos de entidades.
- 3) Identificar los diferentes tipos de relaciones.
- 4) Buscar las cardinalidades adecuadas.
- 5) Identificar los atributos de cada tipo de entidad.
- 6) Identificar las claves de cada tipo de entidad.

Es importante distinguir los tipos de entidades e interrelaciones de los atributos. Los atributos deben ser atómicos y las características del tipo de entidad o interrelación que describan.

## 3. Modelo Relacional

### 3.1. Terminología del modelo relacional.

Cuando hablamos del **modelo relacional** no podemos hacerlo sin nombrar a **Edgar F. Codd**. Él fue la persona que apostó por buscar un modelo que pudiera dar solución a los problemas ocasionados por las BBDD, permitiendo trabajar de forma sencilla sin que estos problemas lo impidieran.

### 3.2. Concepto de relación. Propiedades y relaciones.

El elemento principal de este modelo es la relación que va a utilizarse para representar las diferentes entidades e incluso algunas relaciones de los diagramas entidad- relación. A las filas de una relación se les denomina tuplas. Cada una de las tuplas tiene una serie de atributos con un determinado valor.

The diagram illustrates a relational table with the following structure:

NIF	Nombre	Apellidos	Teléfono	Fecha nacimiento	Fecha alta
74512593P	María	Rodríguez Moreno	635984754	13/07/1980	21/09/2016
48915264M	José	Martín Delgado	650221416	09/02/1975	14/10/2013
21733456Z	Judit	Casas Pérez	679548979	22/06/1963	10/05/2015
32564542F	Marta	Alonso Sánchez	43315642	07/12/1992	22/06/2016

Annotations explain the components of the table:

- Valor**: Points to the value "74512593P" in the first row.
- Atributo**: Points to the header "Nombre" in the second column.
- Tupla**: Points to the entire first row.

*Relación socio.*

### 3.3. Atributos y dominio de los atributos.

Al hablar de atributo tenemos que explicar su dominio.

Si hablamos de atributo, debemos explicar cuál es su dominio. De este modo, el **dominio de un atributo** es el **conjunto de valores posibles que se pueden tomar**. Siempre asociamos la idea de dominio a la de tipo de datos, entre los que podemos diferenciar los siguientes: números enteros y decimales, cadenas de caracteres, fechas y horas, valores lógicos y objetos.

Los datos se organizan en relaciones, y una relación R está definida sobre un conjunto de dominios D<sub>1</sub>, D<sub>2</sub>, ..., D<sub>n</sub> y consta de:

- **Cabecera**: es un conjunto finito de pares (atributo: dominio).

{(A<sub>1</sub>:D<sub>1</sub>), (A<sub>2</sub>:D<sub>2</sub>), ..., (A<sub>n</sub>:D<sub>n</sub>)}  
 Donde cada atributo se corresponde con un único dominio (no al contrario) no habiendo dos atributos que se tengan el mismo nombre.

- **Cuerpo**: conjunto de variables de tuplas.

Cada tupla, es un conjunto de pares atributo: valor.

{(A<sub>1</sub>:V<sub>i1</sub>), (A<sub>2</sub>:V<sub>i2</sub>), ..., (A<sub>n</sub>:V<sub>im</sub>)} con i=1,2, ..., m donde m es la cardinalidad de R. Para cada par, se cumple que V<sub>ij</sub> pertenece a D<sub>j</sub>.

A continuación, vamos a ver aquellos términos más importantes del modelo relacional:

- **Tupla**: cada una de las filas de una tabla.
- **Atributo**: cada columna de las que consta una determinada tabla.
- **Dominio**: el conjunto de valores al que pertenece cada uno de los valores de una columna determinada.
- **Grado**: el número de atributos de una relación.
- **Cardinalidad**: el número de tuplas de una relación.
- **Base de datos relacional**: conjunto de relaciones normalizadas.

Veamos el siguiente ejemplo:

Nombre	Función
Dpto1	Ventas
Dpto2	Estadística
Dpto3	Compras

*Relación departamento.*

Cabecera: {(Nombre: NOMBRE), (Función: FUNCIÓN)}

Cuerpo: una tupla {(Nombre: Dpto1),(Función: Ventas)}

Las **propiedades** que deben cumplir todas las relaciones son las siguientes:

- Cada R tiene que tener un nombre distinto al de las demás.
- Cada R tiene un número fijo de atributos para todas las tuplas.
- Cada atributo tiene un único dominio.
- No importa el orden de los atributos.
- En una R no puede haber dos atributos con igual nombre.
- Valores de atributos atómicos, es decir, en cada tupla un atributo toma un solo valor del dominio.
- Cada tupla es única, distinta a las demás.
- No importa el orden de las tuplas.

### 3.4. Concepto y tipos de clave: candidatas, primarias, ajenas, alternativas

Una **clave** es un **conjunto de atributos que pueden identificar de manera única una entidad**. Es decir, en una relación no puede haber tuplas repetidas, por lo que se tienen que diferenciar unas de otras por algún campo o conjunto de campos. Existen claves simples (atómicas) o compuestas y podemos diferenciar varios tipos:

- **Claves Candidatas:** conjunto de atributos que optan a ser clave principal o primaria.
- **Clave principal/Clave primaria:** atributo o conjunto de atributos que identifican de modo único las tuplas de una relación. Este conjunto de atributos debe de pertenecer al conjunto de claves candidatas del punto anterior y que, por alguna razón, el administrador de la base de datos, lo ha elegido como clave primaria.
- **Claves Alternativas:** conjuntos de atributos perteneciente a las claves candidatas que no son elegidas como clave primaria.
- **Claves Ajenas o foráneas:** atributo o conjunto de atributos de una relación que son clave primaria de otra relación distinta y que por causas del diseño deben estar relacionadas.

### 3.5. Otros conceptos: tupla, grado, cardinalidad, valores nulos, comparación con ficheros.

Los conceptos de tupla, grado, cardinalidad, valores nulos y comparación con ficheros son conceptos base a la hora de entender el modelo relacional, dichos conceptos recogidos en este punto se han definido en los puntos de este tema y anteriores, pero quepa recalcar su importancia.

### 3.6. Reglas de integridad: de entidad y referencial

La **integridad** es uno de los objetivos que deben cumplir los SGBD. Gracias a la integridad de los datos se dificulta la perdida de estos.

Parte de este objetivo puede cumplirse si se cumple la coherencia y veracidad de la información de la BDD. Cabe tener en cuenta que las operaciones de inserción, borrado y modificación de tuplas pueden afectar a la integridad. Si el SGBD no asegura desde un principio la integridad, ésta debe ser garantizada por las aplicaciones.

Para mantener la integridad de la BDD hay que tener en cuenta algunas restricciones como, por ejemplo, cada vez que se inserte o se modifique algún valor del atributo.

Las principales **restricciones** que deben cumplirse en todas las BBDD son:

- **Reglas de la Integridad de la Entidad**
  - Ningún atributo que forma parte de la clave primaria puede ser nulo (Nulo= ausencia de valor, valor desconocido).
  - Se deben realizar comprobaciones en cada inserción y modificación.
- **Reglas de la Integridad Referencial**
  - Los valores de una clave ajena o coinciden con un valor de la clave primaria a la que referencian o son nulos.
  - Un valor nulo en la clave ajena significa que la interrelación entre las entidades es opcional.
  - Si la clave ajena nunca toma valores nulos la interrelación es obligatoria.
  - Para cada clave ajena se debe especificar si debe o no tomar valor nulo y determinar las consecuencias de las operaciones de inserción, borrado y modificación sobre las tuplas de la relación referenciada.
  - Si realizando la BDD llega una petición para realizar una operación ilegal, debemos: rechazarla o aceptarla y conducirla hasta conseguir un estado legal.
  - Debemos seguir una serie de opciones para mantener la integridad referencial.

#### Nulos no permitidos

No se admiten valores nulos de la clave ajena. Tienen que coincidir siempre con un valor de la clave primaria.

### **Restringido**

Un valor de clave primaria no puede ser modificado ni borrado si existe alguna tupla en otra R que lo contenga como clave ajena.

### **Transmisión en cascada**

Si borramos o modificamos una tupla de R que contiene la clave primaria referenciada en otra R diferente por una clave ajena, se producirá el borrado o modificación en cascada de todas las tuplas de la otra R que contengan esa clave como ajena.

### **Puestas a nulos**

El borrado o modificación de una tupla de una R que contiene la clave primaria referenciada en otra R por una clave ajena producirá la puesta a nulo del valor de la clave ajena en todas las tuplas de la otra R que contengan esa clave como ajena.

### **Puesta a un valor por defecto**

El borrado o modificación de una tupla de una R que contiene la clave primaria referenciada en otra R' por una clave ajena, producirá la puesta a un valor por defecto de la clave ajena en todas las tuplas de las R' que contengan esa clave como ajena.

### 3.7. Traducción del modelo entidad-relación al modelo relacional.

La transformación de un esquema del modelo E-R al modelo relacional, se basa fundamentalmente en dos principios básicos:

1. Toda entidad pasa a convertirse en una relación.
2. Toda relación que tiene cardinalidad máxima muchos a muchos (N:M) se transforma en una relación (tabla).

En este proceso de transformación de esquemas casi siempre se pierde semántica ya que el modelo relacional no hace diferencia entre entidades y relaciones.

**El modelo E-R coge más semántica que el relacional**, por lo que podemos utilizar los objetos que nos proporciona el modelo como pueden ser disparadores, procedimientos, etc.

#### a) Transformación de entidades

- Las entidades se convierten en tablas.
- Una tabla siempre se va a llamar de la misma forma que la entidad de la que proviene.

#### b) Transformación de atributos de entidades

- Cada atributo de una entidad se va a transformar en otro atributo de una tabla a la que ha dado lugar la entidad.
- Se van a tener en cuenta:
  - o Los atributos clave pasan a ser clave primaria de la relación.
  - o Los demás atributos van a ser columnas de la tabla, sabiendo que aquellos atributos no optionales se les debe aplicar la restricción *not null* y a las claves alternativas la restricción *unique*.
  - o Para mayor eficiencia, los atributos derivados se pueden transformar en columnas.

#### c) Transformación de relaciones. Tenemos que diferenciar según la cardinalidad que tenga la relación en:

- o **Relaciones “muchos a muchos” (M:N)**: puede transformarse en una tabla que tenga como clave primaria la resultante de la concatenación de las demás claves primarias de las tablas que se han ido creando al transformar las entidades que forman parte de la relación.

- **Relación “uno a muchos” (1:N):** este tipo de relación podemos solucionarlo de dos formas diferentes:
  1. Podemos propagar aquellos atributos elegidos como clave principal de la entidad con cardinalidad a 1 hasta la que vale N.
  2. Se puede transformar la relación en una tabla como si fuera un tipo de relación “muchos a muchos”. Si fuera así, la clave principal de la tabla se va a comportar como clave primaria de la tabla correspondiente a la de cardinalidad M.

Los casos en los que esta opción es conveniente son:

- Cuando el número de ocurrencias de la entidad que se propaga es muy pequeño. Se pueden producir valores nulos, aunque esto no es muy conveniente.
- Cuando se intuye que una determinada relación puede llegar a convertirse en una relación “muchos a muchos”.
- Cuando una relación tiene sus propios atributos y no queremos que se propaguen para que no se pierda semántica.

- **Relaciones “uno a uno”:** cuando se puede generar una nueva tabla como si fuera una relación “muchos a muchos” o, también se puede propagar la clave como si tuviéramos una relación “uno a muchos”.

Para tomar la decisión correcta, podemos diferenciar:

1. Cuando las entidades que forman parte de la relación tienen cardinalidad (0,1) y se puede deducir que no existe relación entre entidades, se genera una nueva tabla, pero como si tuvieran relación “muchos a muchos”.
2. Cuando las entidades que forman parte de la relación tienen cardinalidad (0,1) mientras que la otra es (1,1). Si es así, interesa propagar la clave de (1,1) a la otra y de esta forma, evitar valores nulos.

## 4. Normalización

### 4.1. Concepto de normalización, dependencias funcionales y sus tipos

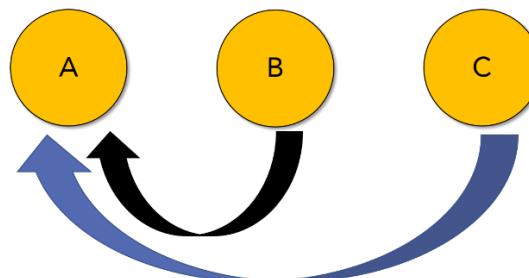
Cuando estamos diseñando una BDD debemos seguir una serie de pasos y, a la hora de finalizar, debemos pasar del modelo entidad-relación al modelo relacional. Esto es lo que se conoce con el nombre de normalización. Cuando diseñamos una BDD o incluso un sistema informático debemos medir su calidad y podemos hacerlo mediante la forma normal.

El **objetivo** de la normalización es:

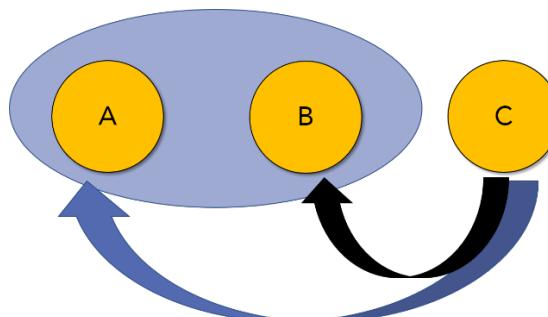
- a) Eliminar la redundancia.
- b) Eliminar la inconsistencia de datos.
- c) Garantizar la integridad referencial.

Antes de profundizar un poco más en las formas normales, acararemos un poco el concepto de dependencia funcional:

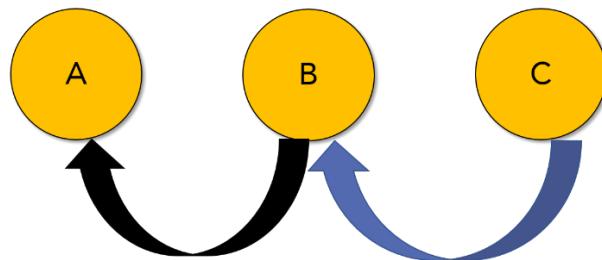
- d) **Dependencia funcional.** Un atributo depende directamente de la clave.



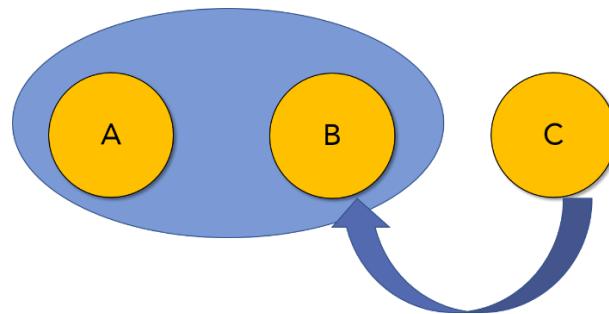
- e) **Dependencia funcional completa.** Un atributo depende de la totalidad de la clave (casos clave compuestos por más de un atributo).



- f) **Dependencia funcional transitiva.** Un atributo depende de otro atributo no clave que presenta una dependencia funcional con la clave.



- g) **Dependencia parcial.** Un atributo no depende de la totalidad de la clave (casos de claves compuestas por más de un atributo) sino que depende funcionalmente de uno de sus atributos.



## 4.2. Primera forma normal (1FN)

No se permite que en una tabla existan atributos que tomen más de un valor. Todos sus valores deben ser atómicos, es decir, univaleados.

The diagram illustrates the decomposition of a table from First Normal Form (1FN) to Second Normal Form (2NF). On the left, a 1FN table contains multiple values for the 'TELÉFONO' column. A large blue arrow points to the right, where the table is decomposed into two 2NF tables. The first table, 'ALUMNOS', contains unique combinations of NIF, NOMBRE, and APELLIDOS. The second table, 'ALUMNOS - TELÉFONO', contains unique combinations of NIF and TELÉFONO.

NIF	NOMBRE	APELLIDOS	TELÉFONO
12345678A	José	Pérez	555332211 666221133
12345678B	María	Martínez	555444555 666555777

NIF	NOMBRE	APELLIDOS	TELÉFONO
12345678A	José	Pérez	555332211
12345678A	José	Pérez	666221133
12345678B	María	Martínez	555444555
12345678B	María	Martínez	666555777

El primer paso es atomizar el campo teléfono como se muestra. Esta solución genera una fuerte redundancia de datos (los campos nombre y apellidos).

Por lo que una solución correcta sería:

ALUMNOS		
NIF	NOMBRE	APELLIDOS
12345678A	José	Pérez
12345678B	María	Martínez

ALUMNOS - TELÉFONO	
NIF	TELÉFONO
12345678A	555332211
12345678A	666221133
12345678B	555444555
12345678B	666555777

La solución óptima pasa por separar los datos en dos tablas diferentes: una para los datos que ya eran atómicos y otra para el campo multivaluado.

### 4.3. Segunda forma normal (2FN)

Para que un atributo se encuentre en la 2FN debe estar en la 1FN y, además, los atributos que no forman parte de la clave tienen una dependencia completa de la clave principal.

CodLibro	CodTienda	Stock	Dirección
12	22	456	C/ Falsa, 123
22	45	567	C/ Elm, 13
44	22	4	C/ Falsa, 123

Como podemos ver, el campo dirección depende de “CodTienda”, pero no de “CodLibro”, por tanto, esta tabla no se encuentra en 2FN.

Para pasar esta tabla a 2FN:

STOCK			DIRECCIONES	
CodLibro	CodTienda	Stock	CodTienda	Dirección
12	22	456	22	C/Falsa, 123
22	45	567	45	C/ Elm, 13
44	22	4	22	C/ Falsa, 123

Podemos ver que la solución que estamos buscando pasa por dividir la información en dos tablas de forma que tengamos una dependencia completa de la clave en ambas.

#### 4.4. Tercera forma normal (3FN)

Decimos que una relación está en 3FN cuando está en 2FN y cuando los atributos que no forman parte de la clave primaria son independientes entre sí, es decir, no presentan dependencias transitivas.

Cod. Emp	Nombre	Apellidos	Fecha Nac	Cod. Dpto	Nombre Dpto
001	Pepe	Pérez	22/12/1958	12	Cuentas
002	María	Pascuález	12/11/1987	11	RRHH

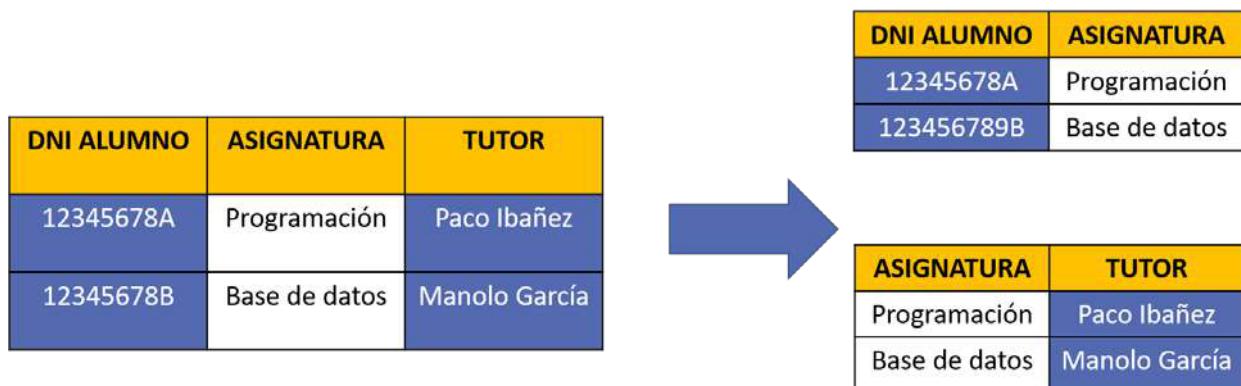
Podemos ver que el campo “Nombre Dpto” depende transitivamente de la clave a través del campo “Cod. Dpto”, por tanto, esta tabla no está en 3FN.

Para llegar a una solución óptima podemos pasar a crear dos tablas, eliminando la información que tenía dependencia transitiva y creando una nueva tabla para ella.

EMPLEADOS					DEPARTAMENTOS	
Cod. Emp	Nombre	Apellidos	Fecha Nac	Cod. Dpto	Cod. Dpto	Nombre Dpto
001	Pepe	Pérez	22/12/1958	12	12	Cuentas
002	María	Pascuález	12/11/1987	11	11	RRHH

#### 4.5. Forma normal Boycce-Codd

La **FNBC** es una versión de la 3FN un poco más estricta. Decimos que una relación está en FNBC cuando está en 3FN y cuando todos sus atributos no clave son clave candidata.



En resumen, para que una tabla se encuentre correctamente normalizada debemos seguir tres principios fundamentales:

1. No pueden existir atributos multivaluados.
2. Todos los atributos deben depender de forma completa y No transitiva de la clave.
3. Si existen claves candidatas compuestas, no deben tener un atributo común.

#### 4.6. Otras formas normales (4FN, 5FN)

Estas formas normales se van a ocupar de las dependencias entre atributos multivaluados.

#### 4.7. Desnormalización

Podemos definir la desnormalización como aquel proceso mediante el cual pretendemos optimizar el desarrollo de una BDD mediante la agregación de aquellos datos redundantes.

## UF2: Lenguajes SQL: DML y DDL

### 1. Lenguajes de las BBDD. SQL

El lenguaje de programación **SQL** es el lenguaje fundamental de los SGBD relacionales. Una de las características principales de este lenguaje es su aspecto declarativo en vez de imperativo., es decir, el programador debe indicar qué hacer y no cómo hacerlo.

Este lenguaje pretende ser lo más natural posible y, por esta razón, se le considera de cuarta generación.

Los elementos que componen el lenguaje SQL son los siguientes:

- a) **DML (*Data Manipulation Language*)**: es el lenguaje que manipula los datos ya creados. Modifica los registros o tuplas de la BDD.
- b) **DDL (*Data Definition Language*)**: permite la creación de la BDD y de las tablas que la componen.
- c) **DCL (*Data Control Language*)**: administra a los usuarios de las BDD, concediendo o denegando los permisos oportunos.
- d) **TCL (*Transaction Control Language*)**: lenguaje que controla el procesamiento de las transacciones de las BDD.

A continuación, para terminar este apartado de introducción al SQL, vamos a nombrar las **normas básicas** para tener en cuenta para diseñar instrucciones en SQL:

1. No se distingue entre mayúsculas y minúsculas.
2. La instrucción en SQL debe terminar con el carácter ";" (punto y coma). Esto se debe a que el compilador está diseñado para que vaya decodificando la instrucción hasta que se encuentre con este carácter que delimita el fin del comando.
3. Antes de finalizar la instrucción, cualquier comando puede ir seguido por: un espacio en blanco o un salto de línea.
4. Se puede tabular la instrucción para clarificar la orden deseada.

## 1.1 Herramientas para gestionar los datos en un SGBDR corporativo

En este tema vamos a desarrollar los distintos lenguajes que pueden intervenir a la hora de detallar el proceso de implantación de una base de datos en un determinado sistema informático.

Existen múltiples herramientas a la hora de crear y manipular una determinada base de datos si disponemos de alguna interfaz gráfica que ayude al DBA (*Database Administrator*). Esta también debe ser capaz de enviar diferentes comandos de administración de forma automática sin necesidad de profundizar con más detalle sobre su sintaxis.

**a) PhpMyAdmin de MySQL:** es una interfaz basada en diferentes páginas web disponibles para MySQL.

Tiene disponibles diferentes opciones que nos permiten visualizar cualquier gestión que realicemos en la BBD como crear, modificar o borrar tablas. Además, permite también importar/exportar información, estadísticas, copias de seguridad, etc.

**b) Oracle Enterprise Manager y Grid Control:** dispone de dos herramientas gráficas con su correspondiente interfaz web. Estas están montadas en un servidor web de Oracle:

- **Enterprise Manager**

Permite manipular cualquier función básica correspondiente a una base de datos. Esta herramienta ya viene incorporada en el software de Oracle.

- **Grid Control**

Gestiona diferentes bases de datos en distintos servidores y ofrece posibilidad de poder consultar el estado y rendimiento de cualquiera de ellas. Se debe instalar a parte del software de Oracle.

**c) DB2 Data Studio:** ofrece la posibilidad de manipular y modificar los objetos de los permisos de una base de datos. Facilita la construcción de consultas SQL. Permite crear Servicios web para poder distribuir datos de consulta.

Para la realización de este módulo, necesitaremos un servidor **Mysql** con el cliente **HeidiSql** para el tratamiento de las bases de datos.



Para continuar vamos a explicar el **proceso de instalación**, de esta manera podremos comenzar a crear y manipular nuestra BDD:

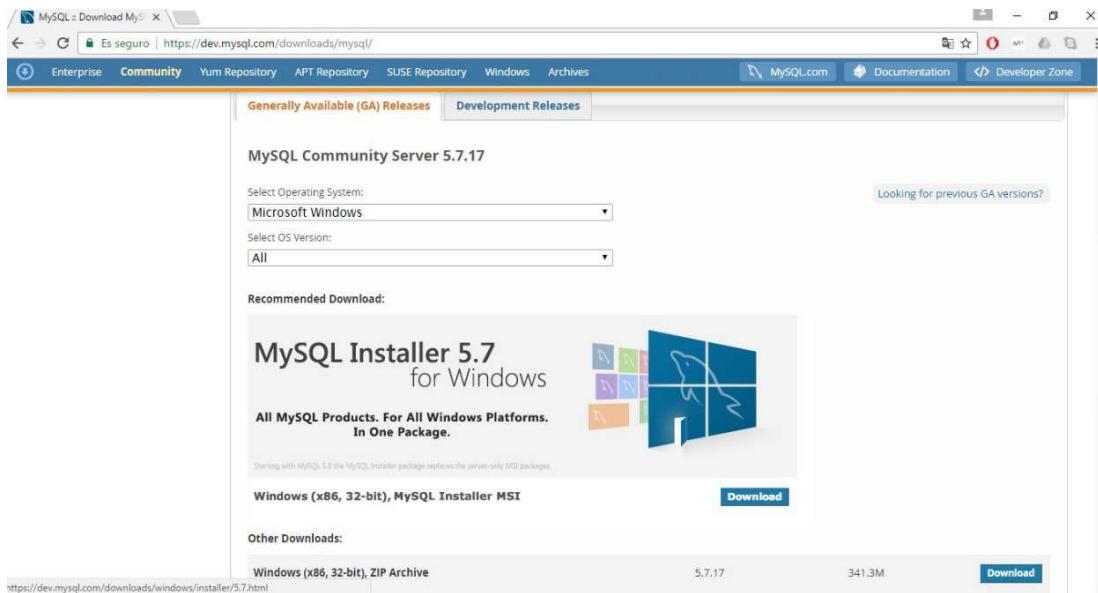
1. Para comenzar, debemos descargar el servidor de BDD, en este caso y para este primer módulo, hemos optado por un gestor de bases de datos orientado a la programación web como es Mysql.

Podemos descargarlo directamente desde su página oficial:

[www.mysql.com](http://www.mysql.com)

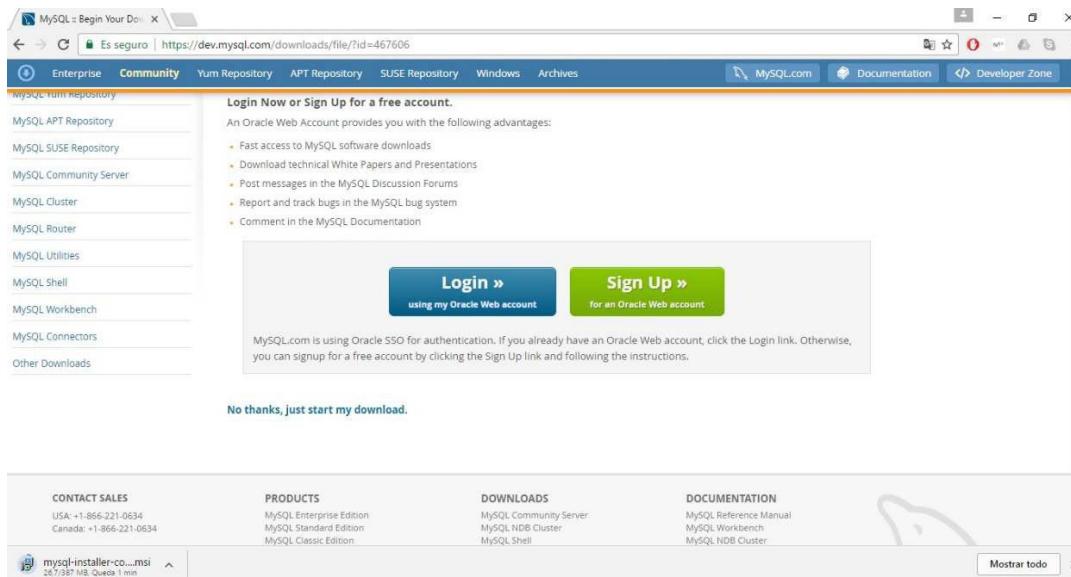
The screenshot shows the MySQL Downloads page. At the top, there's a navigation bar with links for MySQL.com, Downloads (which is highlighted), Documentation, and Developer Zone. Below the navigation, there's a sidebar with links for MySQL on Windows, MySQL Yum Repository, MySQL APT Repository, MySQL SUSE Repository, MySQL Community Server (which is selected and highlighted in blue), MySQL Cluster, MySQL Router, MySQL Utilities, MySQL Shell, MySQL Workbench, MySQL Connectors, and Other Downloads. The main content area is titled "Download MySQL Community Server". It explains that MySQL Community Edition is a freely downloadable version supported by an active community. It notes that MySQL Cluster Community Edition is available separately. There are sections for "Important Platform Support Updates" and "Online Documentation" which lists installation instructions for MySQL 5.7, 5.6, and 5.5 releases. To the right, there's a note about MySQL open source software being provided under the GPL license, and a section for OEMs, ISVs, and VARs to purchase commercial licenses. At the bottom, there's a link to report bugs.

2. Una vez dentro su página web accederemos al apartado “**Descargas**”.

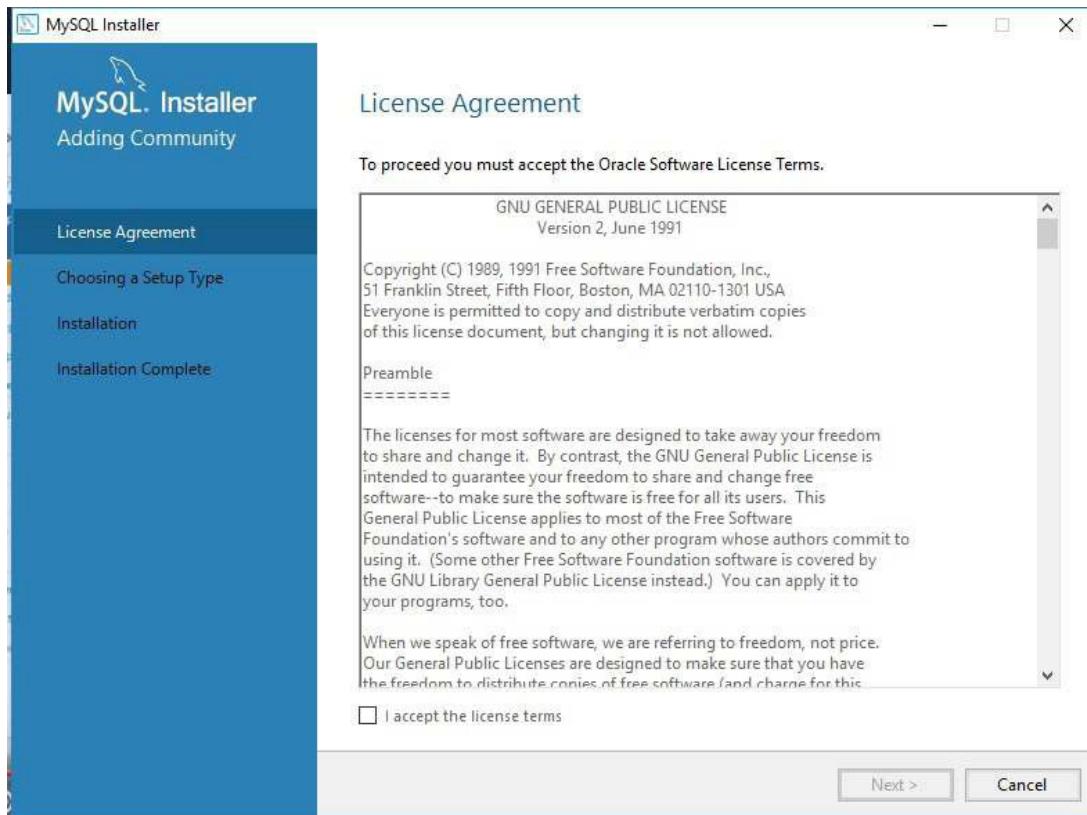


Nos aparecerá una pantalla en la que debemos escoger el sistema operativo sobre el que se va a instalar el servidor de bases de datos, y a continuación, pulsaremos “**Download**”.

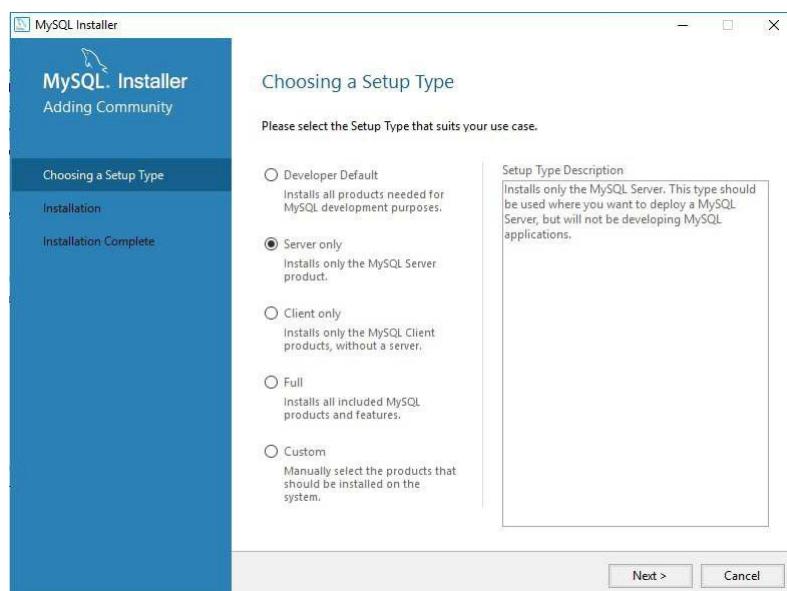
A continuación, la misma instalación, nos dará la opción de escoger entre la versión completa o la **versión reducida**: elegiremos la de menor tamaño ya que, para trabajar el módulo, esta versión cuenta con recursos suficientes.



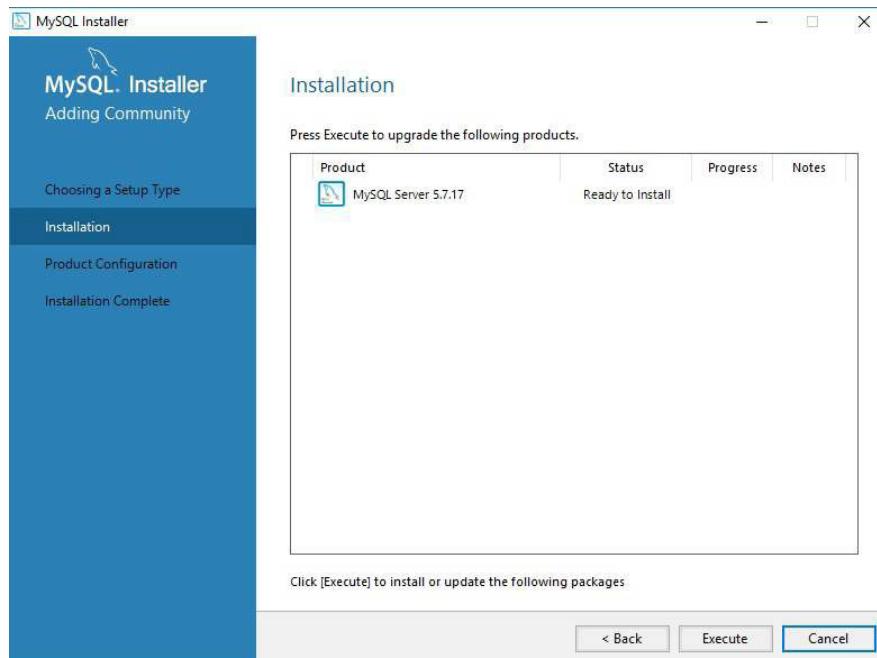
- A continuación, nos pedirá si deseamos registrarnos para comenzar la descarga. Nosotros elegiremos la opción de “**No registrar**” y empezará la descarga.



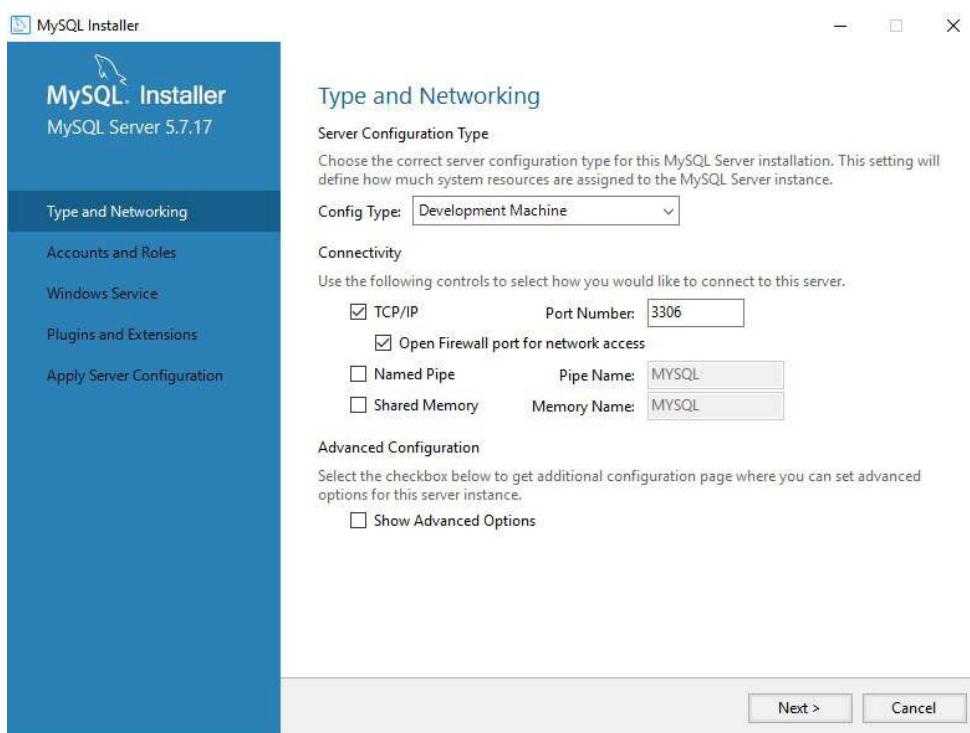
- Para continuar con la instalación aceptaremos las condiciones de la licencia.



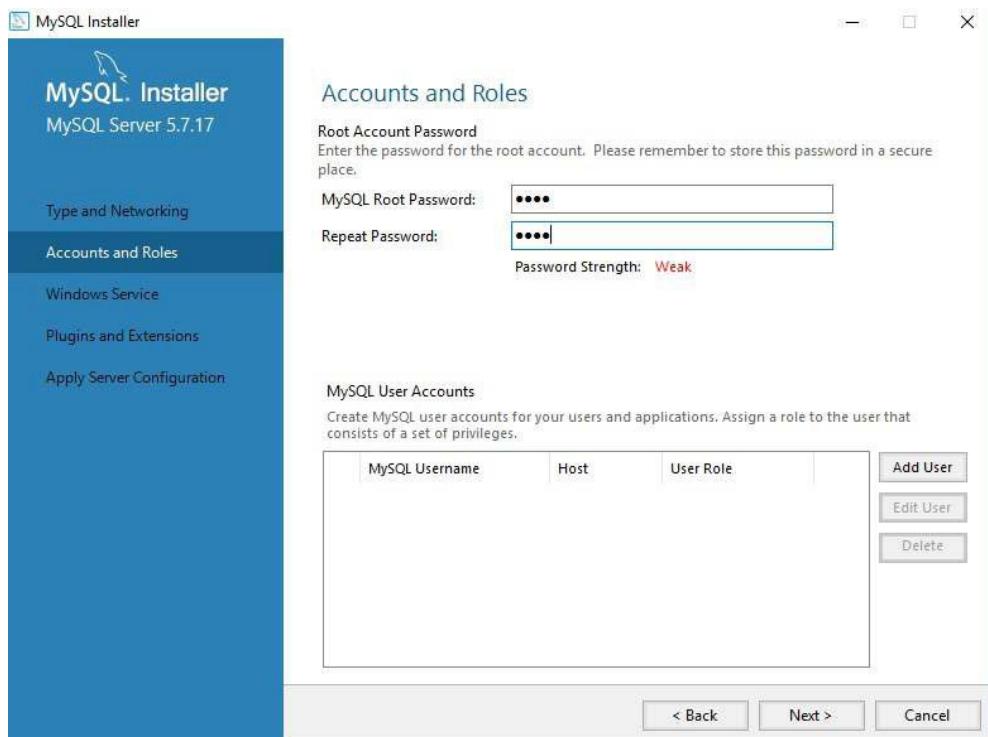
5. Llegado a este punto escogeremos el producto a instalar. Mysql nos dará la opción de obtener tanto el servidor como el cliente, en este caso elegiremos instalar únicamente el servidor en “**Instalar Servidor**”.



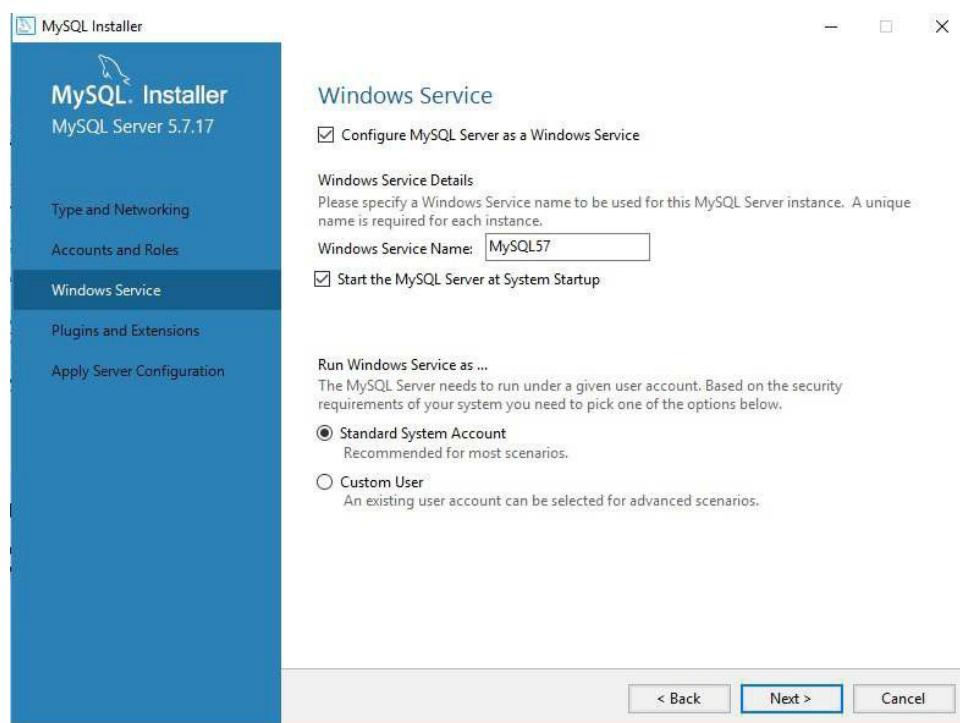
6. Debemos asegurarnos de que la opción escogida es la correcta, para eso compararemos nuestro proceso de instalación y sus opciones con la captura de pantalla proporcionada. Cuando estemos seguros y avancemos se **ejecutará en el producto por el que hemos optado**.



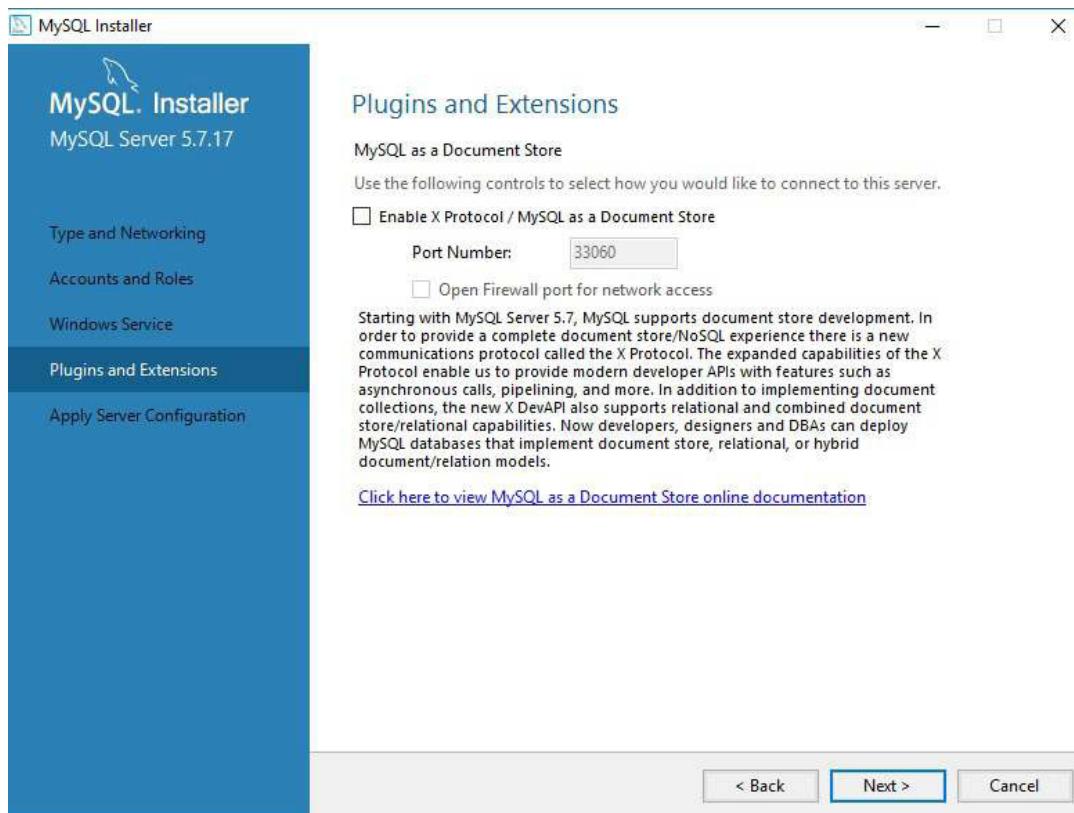
7. Una vez instalado el producto debemos configurarlo. Nos dará la opción de elegir el número del puerto por el que nos vamos a conectar. Por simplicidad en el proceso de instalación, dejamos las características por defecto.



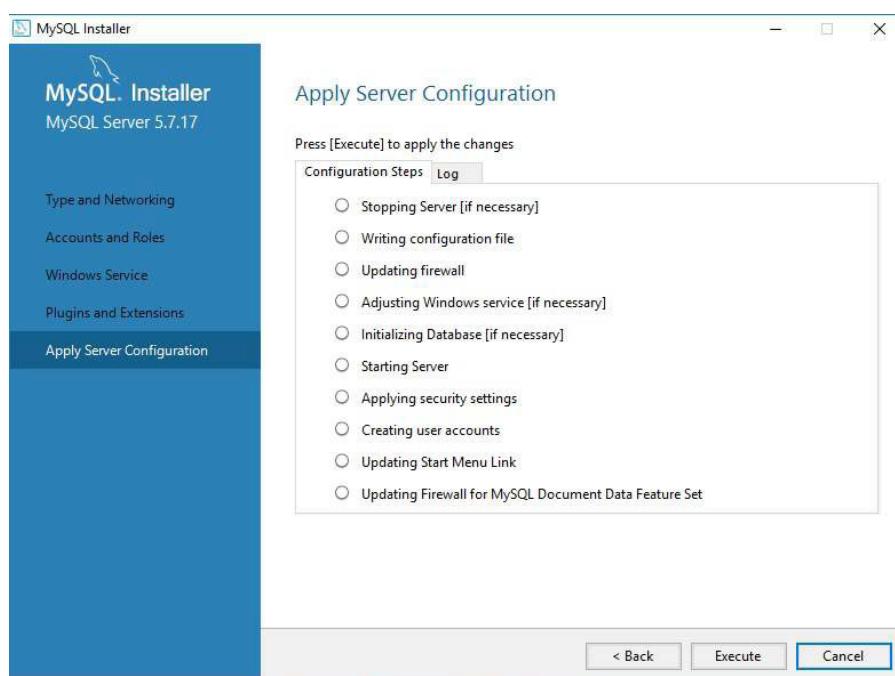
8. Llegados a este punto, el instalador nos pedirá una contraseña para el servidor de la base de datos. Es recomendable que sea fácil de recordar ya que será la que utilicemos en un futuro. Nosotros hemos elegido la palabra “**root**”.



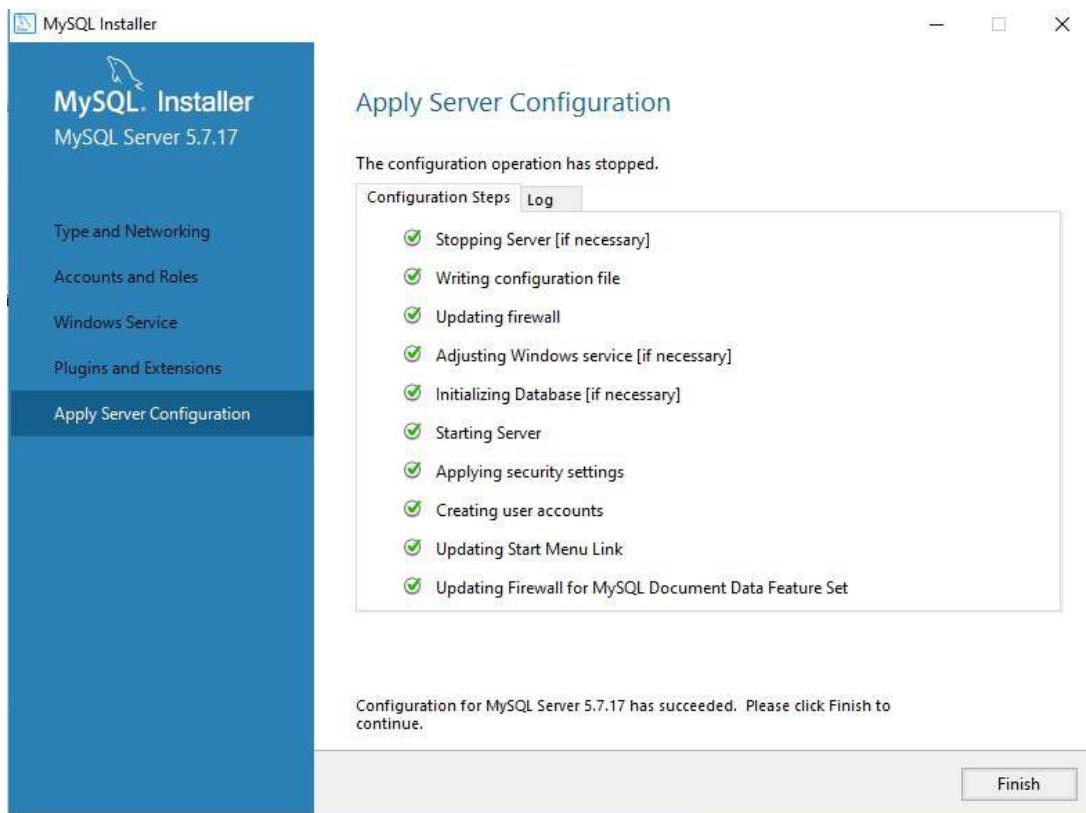
9. Seguiremos el proceso seleccionando el tipo de cuenta que va a crear el servidor y el nombre del servicio. Como en los apartados anteriores, dejaremos la configuración por defecto.



Para todos los procesos siguientes seleccionaremos la configuración por defecto y pulsaremos “**Next**”.



10. Antes de terminar el proceso de instalación, el programa nos mostrará las opciones y ajustes que se van a aplicar una vez pulsemos en “Ejecutar”.



Después de ese paso comprobaremos el éxito del proceso.

## 1.2 Lenguaje de definición de datos (DDL)

Un **DDL** es la parte que realiza la función de definición de datos del SGBD, es decir, es la que se encarga de **definir, modificar y eliminar las estructuras básicas de la BDD**.

Esos objetos pueden ser: bases de datos, tablas, vistas, índices y otros objetos relacionados con la definición de la base de datos. En este apartado solo veremos creación, modificación y eliminación de tablas (con sus respectivas restricciones de campos).

- **Creación de bases de datos**

```
CREATE DATABASE nombre_bd;
```

**Ejemplo:**

```
CREATE DATABASE Centrollerna;
```

- **Creación de tablas dentro de una base de datos**

Para la creación de una tabla tendremos en cuenta el siguiente formato:

```
CREATE TABLE nombre_tabla (
    columna1 tipodato,
    columna2 tipodato,
    columna3 tipodato,
    {RESTRICIONES}
```

**Ejemplo real:**

```
CREATE TABLE TCiudad (
    nCiudadID NUMERIC (4,0) NOT NULL UNIQUE PRIMARY KEY,
    cNombre VARCHAR (40) NOT NULL
);
```

Para la elección del nombre de tabla debemos cumplir las siguientes condiciones:

- Debe identificar el contenido.
- Máximo 30 caracteres.
- No pueden ser palabras reservadas por el SGBD.
- Debe comenzar por una letra.
- Sólo se permiten las letras, los números, el signo de subrayado “\_” y los caracteres “\$” y “#” (aunque no son los más recomendados).
- El nombre de la tabla debe de ser único.

Para la elección del tipo de datos debemos tener en cuenta la siguiente tabla:

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
TEXTO		
Texto de anchura fija	<ul style="list-style-type: none"> <li>• CHARACTER (<a href="#">n</a>)</li> <li>• CHAR (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• CHAR (<a href="#">n</a>)</li> </ul>
Texto de anchura variable	<ul style="list-style-type: none"> <li>• CHARACTER VARYING (<a href="#">n</a>)</li> <li>• VARCHAR (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• VARCHAR2(<a href="#">n</a>)</li> </ul>
Texto de anchura fija para caracteres nacionales	<ul style="list-style-type: none"> <li>• NATIONAL CHARACTER (<a href="#">n</a>)</li> <li>• NATIONAL CHAR (<a href="#">n</a>)</li> <li>• NCHAR (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• NCHAR (<a href="#">n</a>)</li> </ul>
Texto de anchura variable para caracteres nacionales	<ul style="list-style-type: none"> <li>• NATIONAL CHARACTER VARYING (<a href="#">n</a>)</li> <li>• NATIONAL CHAR VARYING (<a href="#">n</a>)</li> <li>• NCHAR VARYING (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• NVARCHAR2(<a href="#">n</a>)</li> </ul>

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
NÚMEROS		
Enteros pequeños (2 bytes)	<ul style="list-style-type: none"> <li>• SMALLINT</li> </ul>	
Enteros normales (4 bytes)	<ul style="list-style-type: none"> <li>• INTEGER</li> <li>• INT</li> </ul>	
Enteros largos (8 bytes)	<ul style="list-style-type: none"> <li>• BIGINT</li> </ul>	
Enteros precisión decimal		<ul style="list-style-type: none"> <li>• NUMBER (<a href="#">n</a>)</li> </ul>
Decimal de coma variable	<ul style="list-style-type: none"> <li>• FLOAT</li> <li>• DOUBLE</li> <li>• DOUBLE PRECISION</li> <li>• REAL</li> </ul>	
Decimal de coma fija	<ul style="list-style-type: none"> <li>• NUMERIC (<a href="#">m,d</a>)</li> <li>• DECIMAL (<a href="#">m,d</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• NUMBER (<a href="#">m,d</a>)</li> </ul>

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
FECHAS		
Fechas	<ul style="list-style-type: none"> <li>• DATE</li> </ul>	<ul style="list-style-type: none"> <li>• DATE</li> </ul>
Fecha y hora	<ul style="list-style-type: none"> <li>• TIMESTAMP</li> </ul>	<ul style="list-style-type: none"> <li>• TIMESTAMP</li> </ul>

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
BOOLEANOS Y BINARIOS		
Lógicos	<ul style="list-style-type: none"> <li>• BOOLEAN</li> <li>• BOOL</li> </ul>	
Binarios	<ul style="list-style-type: none"> <li>• BIT</li> <li>• BIT VARYING (<a href="#">n</a>)</li> <li>• VARBIT (<a href="#">n</a>)</li> </ul>	
DATOS DE GRAN TAMAÑO		
Texto de gran longitud	<ul style="list-style-type: none"> <li>• CHARACTER LARGE OBJECT</li> <li>• CLOB</li> </ul>	<ul style="list-style-type: none"> <li>• LONG (en desuso)</li> <li>• CLOB</li> </ul>
Binario de gran longitud	<ul style="list-style-type: none"> <li>• BINARY LARGE OBJECT</li> <li>• BLOB</li> </ul>	<ul style="list-style-type: none"> <li>• RAW (en desuso)</li> <li>• LONG RAW (en desuso)</li> <li>• BLOB</li> </ul>

- **Borrado de las tablas**

El borrado de tablas se realiza mediante la siguiente orden:

```
DROP TABLE tabla [CASCADE CONSTRAINT];
```

**Ejemplo real:**

```
DROP DATABASE Centrollerna;
```

Al eliminar una tabla también desaparecerán sus datos. Si además añadimos la cláusula de eliminación en cascada (*Cascade Constraint*), también se borrarán las restricciones de integridad referencial que influya a la clave principal de la tabla a borrar.

Una operación de borrado que es **irreversible**. Por ese motivo debemos estar seguros antes de llevarla a cabo.

- **Modificación de la tabla**

Esta operación de modificación nos permite: añadir, modificar o eliminar columnas y activar o desactivar restricciones.

En el siguiente formato podemos ver todas las opciones de esta cláusula:

```
ALTER TABLE nombre_tabla {  
    [ADD (columna)]  
    [MODIFY/CHANGE (colum [....])]  
    [DROP COLUMN (colum....)]  
    [ADD CONSTRAINT restricción]  
    [DROP CONSTRAINT restricción]  
    [DISABLE CONSTRAINT restricción]  
    [ENABLE CONSTRAINT restricción]  
};
```

A continuación, veremos en detalle cada una de las opciones de esta sentencia:

#### **Añadir, modificar o eliminar columnas**

- **ADD:**
  - Si la columna no tiene restricción NOT NULL se añade sin más.
  - Si tiene restricción NOT NULL: se añade, se le asigna un valor y por último se le asigna la restricción.
- **MODIFY/CHANGE:**
  - Se puede aumentar o disminuir la longitud de la columna.
  - Se puede aumentar o disminuir el valor de los decimales del campo.
- **DROP:**
  - No se pueden eliminar todos los campos de una tabla.
  - No se pueden borrar claves primarias referenciadas por una clave ajena de otra tabla.

- **Restricciones**

Las restricciones en las bases de datos hacen que las estructuras sean más eficaces y, por tanto, tengamos menos trabajo a la hora de manipular los datos.

Hay distintos tipos de restricciones:

- Clave primaria (PRIMARY KEY)
- Clave ajena (FOREIGN KEY)
- Columnas no nulas (NOT NULL)
- Valores por defecto (DEFAULT)
- Verificador de condiciones (CHECK)
- Valor único (UNIQUE)

A continuación, vamos a ver cómo podemos definir, manipular o eliminar estas restricciones. Cuando creamos una tabla podemos crear restricciones para sus campos de dos formas distintas:

- **En la propia definición del campo**

```
CREATE TABLE nombre_tabla (
nomcolum_1 TIPO_DATO() TIPO_RESTRICCION,
..... );
```

- **Al final de la creación de la tabla**

```
CREATE TABLE nombre_tabla(column1 tipo_dato(),
........................
[CONSTRAINT nombre_restriccion] UNIQUE (columna [columna])
,.....);
```

Para terminar, vamos a ver unos ejemplos de **cómo serían las distintas restricciones** que podemos definir en el lenguaje SQL mediante **SGBD MySql**:

**1. CREATE TABLE empleados(**

```
dni INT PRIMARY KEY,
nombre TEXT(15),
apellidos TEXT(30),
cod_postal INT CONSTRAINT cod REFERENCES codigos_postales);
```

**Explicación:** en esta tabla la restricción de clave primaria se aplica al campo "DNI", y la restricción de clave foránea al campo "COD\_POSTAL" -referido a una supuesta tabla que se llama "CODIGOS\_POSTALES"-.

**2. CREATE TABLE empleados(**

```
dni INT PRIMARY KEY,
nombre TEXT(15) NOT NULL,
apellidos TEXT(30),
cod_postal INT CONSTRAINT cod REFERENCES codigos_postales,
salario CURRENCY NOT NULL,
CONSTRAINT cf_cp FOREIGN KEY(cod)REFERENCES codigos_postales);
```

**Explicación:** en este caso hemos creado la misma tabla que en el ejemplo anterior, pero hemos añadido restricciones de forma diferente. Tanto la clave primaria como los campos obligatorios (NOT NULL) se implementan a nivel de campo, pero la restricción de la clave foránea se está creando a nivel de tabla, donde iniciamos con la palabra reservada restricción (CONSTRAINT) y, a continuación, un nombre que le damos a la restricción (cf\_cp). Seguidamente nos encontramos con el tipo de restricción (FOREIGN KEY) y como es una clave ajena, entonces le tenemos que indicar a qué clave de otra tabla se refiere (REFERENCES).

También podemos hacer uso de las restricciones en las sentencias ALTER para añadir, modificar o eliminar alguna restricción:

**3. ALTER TABLE empleados**

```
ADD COLUMN cod_oficina INT;
```

*Explicación:* solamente estamos añadiendo el campo código de oficina a la tabla empleados.

**4. ALTER TABLE empleados**

```
ADD CONSTRAINT cod_oficina  
FOREIGN KEY (código)  
REFERENCES oficinas;
```

*Explicación:* en este caso podemos ver cómo se modifica una tabla añadiéndole una restricción a un campo existente, ya que ahora es una clave foránea de una clave principal (código) de la tabla oficinas.

**5. ALTER TABLE empleados**

```
ADD CONSTRAINT u_oficina UNIQUE (cod_oficina);
```

*Explicación:* en este caso, además de que el código de la oficina es clave ajena, le estamos obligando a que su valor sea único, es decir, que en cada oficina solo puede trabajar un empleado.

Cuando trabajamos con sentencias en SQL: para diferencias es mejor utilizar las cláusulas SQL de los nombres que el administrador de campos y restricciones. Normalmente se emplea las mayúsculas para las sentencias y para las palabras reservadas, y las minúsculas para los nombres definidos por el usuario.

### 1.3 Lenguaje de manipulación de datos (DML)

El **DML** (*Data Manipulation Language*) es un lenguaje que ofrece la posibilidad de acceder a la información contenida en una base de datos.

Gracias a este lenguaje tenemos la posibilidad de introducir nueva información, modificarla o incluso borrarla. Su función más importante es llevar a cabo **consultas** sobre una BDD, aunque también tiene implementadas operaciones como la inserción, modificación o eliminación de registros de una tabla.

Las **sentencias** que forman este lenguaje son:

#### **SELECT, INSERT, UPDATE y DELETE**

- **Construcción de sentencias de inserción**

La inserción de datos es una operación obligatoria en las BBDD. Cronológicamente es la primera operación perteneciente a esta parte del lenguaje SQL ya que, sin tener datos almacenados en las relaciones o tablas, no podemos modificar, consultar y borrar.

La sintaxis se desarrollaría de la siguiente forma:

```
INSERT INTO nombre_tabla (columna1, columna2,...)  
VALUES (valor1, valor2,...);
```

Existe otra posibilidad a la hora de introducir valores a todas las columnas de una tabla. No hace falta especificar el nombre de las columnas, como en el ejemplo anterior. El orden de los valores debe de coincidir con el orden de los campos creados en la tabla.

```
INSERT INTO nombre_tabla  
VALUES (valor1, valor2, valor3, ...);
```

- **Construcción de sentencias de modificación**

La sentencia de actualización o modificación se debe hacer sobre datos ya insertados en la tabla.

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2,...
WHERE condicion;
```

Debemos tener cuidado a la hora de utilizar esta sentencia, ya que en la cláusula “Where” tenemos que introducir la condición que deben cumplir los datos para actualizarse. Si omitimos esta cláusula, entonces se van a modificar todos los registros de la tabla.

- **Construcción de sentencias de eliminación**

Lo utilizamos para eliminar registros de una tabla que cumpla una condición. Como en el anterior apartado, si omitimos la cláusula “Where”, se borrará todas las tuplas de la tabla.

```
DELETE FROM nombre_tabla
WHERE condicion;
```

- **Construcción de consultas. La selección simple**

La operación de consultar una BDD se encuentra entre las más utilizadas, ya que es la única forma de sacar partido a los datos que tenemos almacenados en las tablas.

Una consulta en SQL tiene básicamente el siguiente formato:

```
SELECT columna1, columna2,...
FROM nombre_tabla;
```

A medida que vayamos avanzando en los distintos apartados veremos la posibilidad de ampliar esta cláusula.

A continuación vamos a explicar las distintas cláusulas que tiene la sentencia “SELECT”. En la primera parte de la sentencia, en la parte de “SELECT”, tendremos que indicar el nombre de los campos de los cuales deseamos tener información.

Dentro de “FROM” pondremos el nombre de la tabla que interviene en la consulta y, por tanto, los campos anteriores deben pertenecer a dicha tabla.

- **Construcción de consultas de selección con restricción y ordenación**

Este primer ejemplo que vamos a ver es una consulta con restricción en la que tenemos que indicar la condición que deben cumplir los campos para ser devueltos. Esta condición se indicará con la cláusula “**WHERE**”.

```
SELECT columna1, columna2, ...
FROM nombre_tabla
WHERE condicion;
```

La cláusula “**ORDER BY**” se usa para ordenar los datos resultantes de una consulta, se puede hacer de manera ascendente o bien descendente. Su sintaxis sería:

```
SELECT columna1, columna2, ...
FROM nombre_tabla
ORDER BY columna1, columna2, ... ASC|DESC;
```

El valor por defecto sería “Ordenar de manera ascendente”, por tanto, no habría que indicarlo.

En las dos cláusulas que hemos visto en este apartado se pueden complementar, es decir, se les puede añadir una consulta con restricción y que sus valores salgan ordenados.

```
SELECT columna1, columna2, ...
FROM nombre_tabla
WHERE condicion
ORDER BY columna1, columna2, ... ASC|DESC;
```

- **Construcción de consultas de selección utilizando cláusulas del lenguaje para la agrupación**

La sentencia “**GROUP BY**” se utiliza para agrupar por algún campo de la tabla. Se puede usar conjuntamente con una función de agregación, aunque no es obligatorio:

- **COUNT**: devuelve la cantidad de registros.
- **MAX**: devuelve el valor máximo de los registros en el campo seleccionado.
- **MIN**: devuelve el valor mínimo de los registros en el campo seleccionado.
- **SUM**: devuelve la suma de los registros en el campo escogido.
- **AVG**: retorna la media aritmética de los valores del campo.

La sintaxis se desarrollaría de la siguiente manera:

```
SELECT columna1,funcion_agregacion(nombre_columna)
FROM nombre_tabla
WHERE condicion
GROUP BY nombre_columna;
```

- **Construcción de consultas utilizando las funciones añadidas del lenguaje tratando los valores nulos**

Un campo con valor “**NULL**” contiene un valor nulo, es decir, sin valor. En este apartado vamos a utilizar las cláusulas “**IS NULL**” o “**IS NOT NULL**”, ya que no es posible utilizar operadores de comparación con valores “**NULL**”.

Veamos la sintaxis de una consulta evaluando valores nulos:

```
SELECT columna1,columna2....
FROM nombre_tabla
WHERE columna IS NULL / IS NOT NULL
```

- **Construcción de consultas para consultar más de una tabla**

El apartado de las consultas se puede desarrollar ampliamente debido a la existencia de numerosas sentencias que podemos utilizar al diseñarlas. El uso de más de una tabla en una consulta hace que esta operación sea mucho más completa. La sintaxis para realizar una consulta donde intervenga más de una tabla es la siguiente:

```
SELECT tabla.columna1,tabla.columna2....  
FROM nombre_tabla1, nombre_tabla2  
WHERE tabla1.campo_clave=tabla2.campo_clave_ajena;
```

Por ejemplo, si quisiéramos saber el departamento de un empleado llamado “Juan Pérez” lo desarrollaríamos de la siguiente manera:

```
SELECT e.nombre, e.apellidos, d.nombre  
FROM empleado e, departamento d  
WHERE d.cod=e.dep AND e.nombre="JUAN" AND e.apellido= "PEREZ";
```

- **Construcción de subconsultas**

La construcción de subconsultas se realiza cuando queremos que sobre una consulta intervengan más de dos tablas. Para diseñar una consulta con varias tablas a la vez podemos combinar consultas sencillas entre ellas, siempre que tengamos un campo que pueda unir ambas consultas.

En el ejemplo siguiente podemos ver una subconsulta donde deseamos tener conocimiento de los códigos de los libros alquilados por la socia “Paula Sanz González”:

```
SELECT cSignatura  
FROM TPrestamo  
WHERE Cnif = (SELECT Cnif  
FROM TSocio  
WHERE cApellidos = "Sanz González"  
AND cNombre = "Paula");
```

## 1.4 Extensiones y otras cláusulas del lenguaje

A medida que vamos avanzando en el lenguaje SQL vamos introduciendo más sentencias que nos ayudarán a implementar consultas muy completas que satisfagan nuestras necesidades con respecto a las bases de datos.

A continuación vamos a ver la cláusula “UNION” que, como su nombre indica, nos ayuda a unir el resultado de dos o más consultas.

Presenta el siguiente formato:

```
SELECT nombre_columna(s) FROM tabla  
UNION  
SELECT nombre_columna(s) FROM tabla2;
```

Para que esta operación se pueda llevar a cabo de una forma satisfactoria debe cumplir las siguientes condiciones:

1. Las consultas, de forma individual, deben tener el mismo número de campos y con los mismos tipos de datos representados en el mismo orden.
2. Por defecto, este operador solo selecciona para sus resultados a los valores distintos de los campos que lo forman. En el caso de desear valores duplicados, tendríamos que añadir la opción “UNION ALL”.

### Ejemplo:

```
SELECT * FROM ALUMNOS WHERE FECHA_NACIMIENTO<='1970-01-01'  
UNION  
SELECT * FROM ALUMNOS WHERE FECHA_NACIMIENTO>='1980-01-01';
```

De igual manera que hay una cláusula para la unión, hay una sentencia para la diferencia. “EXCEPT” o “INTERSECT”. Tiene el mismo formato que la sentencia anterior, es decir, la diferencia de los resultados de dos consultas.

```
SELECT nombre_columna(s) FROM tabla  
INTERSECT  
SELECT nombre_columna(s) FROM tabla2;
```

La primera consulta debe tener mayor número de resultados que la segunda. Además, también debe cumplir con las condiciones enumeradas en la sentencia anterior.

**Ejemplo:**

```
SELECT productID  
FROM production.product  
INTERSECT  
SELECT productID  
FROM production.workorder
```

En el siguiente apartado vamos a ver los “**INNER JOIN**” junto con sus variantes “LEFT” y “RIGHT JOIN”. El objetivo de todas las sentencias “JOIN” es la de calcular la intersección entre dos tablas o relaciones.

Es una segunda forma de realizar consultas donde intervengan varias tablas.

Su formato es el siguiente:

```
SELECT columna(s)
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna=tabla2.columna;
```

Una variedad de esta sentencia es “RIGTH JOIN”. Esta devuelve las tuplas comunes a las dos tablas y no comunes de la segunda tabla. Mientras que la cláusula “LEFT JOIN” devuelve los comunes y no comunes de la primera tabla.

A continuación, vamos a explicar las sentencias anteriores con un ejemplo:

```
SELECT columna(s)
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna=tabla2.columna;
```

```
SELECT columna(s)
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna=tabla2.columna;
```

Para terminar, vamos a enumerar una serie de cláusulas relevantes a la hora de diseñar una consulta:

- **DISTINCT**: suele indicarse en la parte de “SELECT” para no devolver valores repetidos.

```
SELECT DISTINCT columna1, columna2, ...
FROM tabla1;
```

Ejemplo para seleccionar solo los valores DISTINCT de la columna *Pais* en la tabla *Consumidores*.

```
SELECT DISTINCT Pais FROM Consumidores;
```

- **HAVING**: cláusula que la tenemos en cuenta a la hora de agrupar por alguna condición.

```
SELECT columna, funcion_agregacion(columna)
FROM tabla
WHERE condicion
GROUP BY columna
HAVING condicion_funcion_agregacion;
```

**Ejemplo:**

```
SELECT COUNT(idConsumidor), Pais
FROM Consumidores
GROUP BY Pais
HAVING COUNT(idConsumidor) > 5;
```

## 1.5 Herramientas de la BDD para optimizar consultas

Cuando vayamos a desarrollar una base de datos es fundamental saber qué finalidad va a tener, organizar toda la información de la que dispongamos, definir las relaciones entre las diferentes tablas y elementos que sean necesarias y, finalmente, normalizarla. De esta forma, conseguiremos que la información no esté duplicada y así poder desarrollar una metodología eficiente de la información para un futuro.

Cuando ya la tengamos diseñada, la optimizaremos siguiendo una serie de recomendaciones:

- Eliminar las tablas que hemos creados previamente y que no utilizamos.
- Optimizar los índices de las tablas para asegurarnos de su correcto funcionamiento.
- No dejar consultas abiertas que puedan ralentizarnos el trabajo.
- No almacenar imágenes en la BDD, es mucho mejor referenciar la ruta.
- Nombres simples en claves y campos de las tablas para facilitar el trabajo.

## 2. Estrategias para el control de las transacciones y de la concurrencia

### 2.1 Concepto de integridad

La **integridad** es una pieza fundamental de las bases de datos y se encarga de que los datos que la componen sean lo más correctos posibles. Estos datos almacenados en la base de datos deben cumplir una serie de restricciones con el objetivo de facilitar el trabajo del usuario en cuanto a la manipulación de datos de las BDD.

Aunque es una pieza fundamental puede ocasionar una serie de problemas. Si, por ejemplo, borramos un registro de nuestra tabla principal que, a su vez está relacionado con algunos registros de otra tabla secundaria, va a provocar un error al detectar un fallo de integridad.

Existen claves secundarias que se refieren a una clave principal que ya no está. Por este motivo se pueden aplicar una serie de soluciones que se añaden detrás de la cláusula “**REFERENCES**” como:

- e) **ON DELETE SET NULL**: asigna valores nulos a aquellas claves secundarias que estén relacionadas con la que se ha borrado.
- f) **ON DELETE CASCADE**: elimina aquellos registros que tienen su clave secundaria idéntica a la del registro que se ha eliminado.
- g) **ON DELETE SET DEFAULT**: sitúa en el registro relacionado un valor asignado por defecto en la columna relacionada.
- h) **ON DELETE NOTHING**: no hace ningún cambio.

**Podemos sustituir la palabra “DELETE” por “UPDATE”. Así, el funcionamiento, se va a mencionar cada vez que se produzca algún cambio en la tabla principal.**

Existen, además, reglas de integridad que debemos controlar si nos encontramos con violaciones de la integridad.

Estas reglas se dividen en dos principales:

- **Reglas de integridad de dominios**: si se le asigna un valor a un atributo sin saber la relación que éste tiene con los demás que forman la BDD.
- **Reglas de integridad de relaciones**: cuando se admite una tupla dada para ser insertada o bien cuando se van a relacionar varias tuplas.

## 2.2 Concepto de transacción. Control

Cuando hablamos de una **transacción** nos referimos a un conjunto de diferentes acciones capaces de realizar transformaciones sobre los estados de un sistema conservando su integridad. Una transacción puede ser cualquier tipo de operación atómica que se realice con éxito.

Estas acciones que se van a realizar son independientes las unas de las otras pero relacionadas. De esta manera, inicialmente se comienza abriendo la transacción y, seguidamente, si todas estas acciones se ejecutan de forma correcta, se confirma y se cierra la transacción.

Sin embargo, si se observa cualquier tipo de error en ellas la transacción se deshace. De esta forma se tiene siempre en cuenta la integridad de los datos.

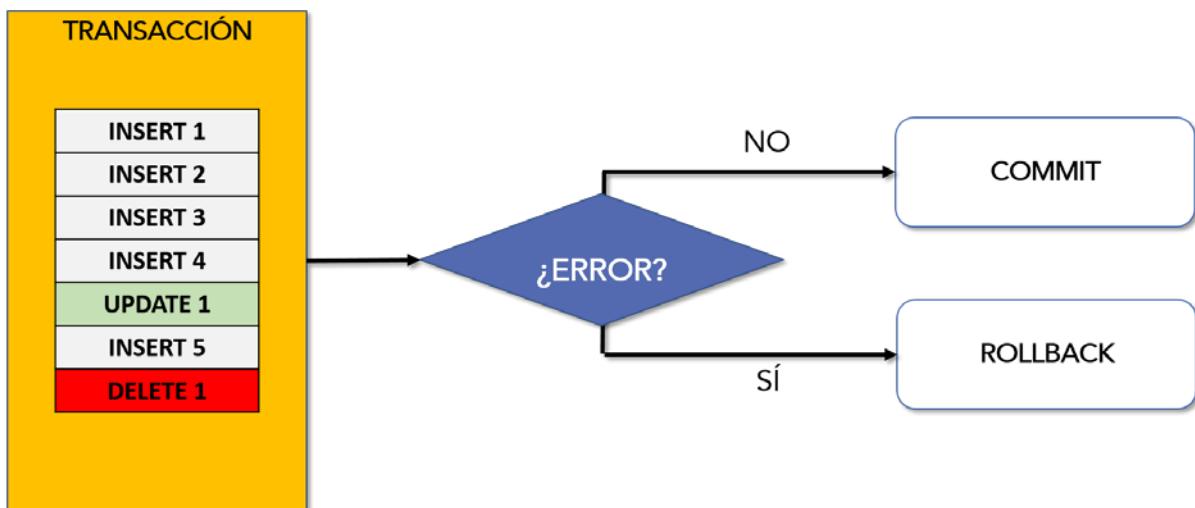
Un buen ejemplo sería una transferencia de dinero de una cuenta a otra: débito y crédito.

## 2.3 Propiedades de las transacciones: atomicidad, consistencia, aislamiento y permanencia

Existen una serie de propiedades de las transacciones que debemos conocer:

- **Atomicidad (Atomicity)**: actúa como un proceso atómico, es decir, o todo (modificación, agregación o borrado) se realiza con éxito, o nada. Basta con que falle una mínima parte para que la operación no sea satisfactoria.
- **Consistencia (Consistency)**: cuando se ejecuta la transacción, el sistema debe pasar de un estado consistente a otro que también lo sea pese a los cambios que se han realizado.
- **Aislamiento (Isolation)**: cada transacción debe actuar de forma secuencial.
- **Permanencia (Durability)**: todos los cambios que se hayan producido cuando se realiza una transacción no se pierden, sino que permanecen.

## 2.4 Estados de una transacción: activa, parcialmente comprometida, fallida, abortada y comprometida



En el dibujo podemos apreciar muy bien su funcionamiento: se realizan una serie de instrucciones en la que la última es una operación de borrado (“**DELETE**”). Si todo se ha realizado sin problema, “commit” (validar) y finaliza la transacción.

Pero si se detecta algún error, “rollback” (cancelar) que elimina los cambios anteriores y vuelve a empezar.

## 2.5 Problemas derivados de la ejecución concurrente de transacciones

Anteriormente hemos visto el conjunto de normas que deben cumplir las transacciones. Además, también es importante que nos detengamos un poco ante los problemas más frecuentes que pueden ocasionar.

Uno de los principales es el que se ocasiona cuando dos transacciones quieren acceder al mismo dato de manera simultánea, le llamamos problema de concurrencia. Cuando nos encontramos ante un caso así podemos diferenciar entre **tres tipos diferentes de problemas**:

- ***Dirty read*** (lectura sucia): cuando una transacción consulta datos escritos de otra que aún no ha sido confirmada.
- ***Nonrepeatable read*** (lectura irrepetible): cuando una transacción vuelve a hacer una lectura de unos datos que ya había leído y comprueba entonces que han sido modificados en alguna transacción.
- ***Phantom read*** (lectura fantasma): cuando una transacción realiza una consulta y encuentra datos que antes eran inexistentes. Alguna transacción los ha insertado.

## 2.6 Control de concurrencia: técnicas optimistas y pesimistas

Uno de los principales problemas que se ocasiona en las transacciones de los datos de una BDD es que se pida acceso a un mismo dato des de dos lugares distintos. Es en ese momento en el que se precisa un **control de concurrencia** para darle solución.

El encargado de este control de concurrencia es el **planificador**, este va a realizar diferentes esquemas para que las transacciones no se solapen entre ellas.

Siempre es conveniente que el planificador no realice ningún cambio en el sistema, tanto si las transacciones se ejecutan de forma concurrente como si lo hacen una detrás de otra.

Veamos qué tipo de técnicas podemos utilizar para evitar este tipo de problemas:

- **Técnicas pesimistas:**

- **Técnicas de bloqueo (*locks*)**

Su tarea principal es poder bloquear aquellos datos para que no se acceda a ellos desde diferentes transacciones (sincronizar el acceso).

Para ello va a hacer uso del cerrojo (**lock**) que va a actuar como una variable para poder controlar el estado de los datos según las operaciones permitidas. Esto conlleva un riesgo importante como es el bloqueo (**deadlock**).

Aunque, si bien es cierto, estos cerrojos no nos garantizan la serialibilidad por sí solos y necesitan el uso de un protocolo para que podemos tener un control del posicionamiento de aquellas operaciones que se hayan realizado dentro de una transacción.

Uno de los más utilizados es el protocolo de bloqueo en dos fases. En la primera fase o de crecimiento es donde van a solicitarse los *locks* mientras que, en la segunda -denominada devolución-, va a ser donde se realizan los *unlocks*.

Este protocolo sí garantiza la serialibilidad, aunque no libera ningún cerrojo desde que comienza hasta que finaliza.

- **Técnicas de marcas de tiempo (*time-stamping*)**

Las marcas de tiempo se utilizan para que sólo exista un único identificador para cada transacción. Estas marcas deben ir en orden para poder el acceso a los diferentes datos sin que estos se solapen.

- **Técnicas optimistas:**

También conocidas como técnicas de validación o de certificación. Estas técnicas no llevan impuestas ninguna restricción específica ni ningún bloqueo. Aunque, al final, hacen una comprobación de tres fases diferentes que se pueden dar: lectura, validación y escritura.

Son bastante adecuadas cuando existen pocas transacciones, así hay menos operaciones.

A continuación, vamos a ver una técnica bastante utilizada que asigna marcas de tiempo de las transacciones, tiempo inicial y final de algunas de las fases.

Debemos apuntar que, en la fase de validación, se va a comprobar que esta no se solapa con ninguna otra transacción que esté confirmada o en fase de validación.

## 2.7 Recuperación ante errores. Mecanismos para deshacer transacciones

Existen diferentes tipos de fallos que nos podemos encontrar:

1. Fallo informático.
2. Error del sistema o transacción: división por cero, errores de algún parámetro, de programación, etc.
3. Errores locales: datos que no se encuentran en una transacción, saldo insuficiente, etc.
4. Control de concurrencia: transacciones que no siguen adelante para garantizar la serialidad.
5. Fallo del disco (lectura/escritura).
6. Problemas físicos.

De hecho, podemos agrupar estos fallos dentro de dos grupos:

- Fallos que pierden contenido de la memoria estable como, por ejemplo, los discos.
- Fallos que pierden contenido de la memoria volátil (memoria principal).

El **mecanismo de recuperación** que tomemos ante cualquier error va a ser el encargado de **restablecer la base de datos** al estado anterior al fallo. Este mecanismo, además, debe reducir el tiempo de uso de la base de datos después de que haya habido un error.

Existen diferentes tipos de **estrategias de recuperación** que podemos llevar a cabo:

- Cuando **se daña una parte de la base de datos**: se puede restaurar una copia previa al fallo y reconstruir un nuevo estado volviendo a realizar las operaciones que ya están almacenadas.
- Cuando la base de datos **no parece dañada, pero no responde**: se pueden realizar las operaciones que nos han llevado a esta situación siguiendo un orden inverso. De esta forma comprobaremos qué acción ha producido el fallo.

### 3. Lenguajes de las BBDD para la creación de su estructura

#### 3.1 Vistas y otras extensiones del lenguaje

Cuando hablamos de vistas nos estamos refiriendo a almacenar el resultado de una consulta sobre una o varias tablas en una estructura. Esta vista se podrá tratar, en un futuro, como si fuera una tabla. Podemos diferenciar entre dos tipos de vistas:

- **Simples:** cuando está formada por una única tabla y no tiene ninguna función de agrupación. Permite operaciones DML.
- **Complejas:** cuando está formada por más de una tabla y sí utiliza funciones de agrupación. No permite operaciones DML.

##### - Creación

###### Sintaxis para crear una tabla

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW nombre_vista [lista_columnas]
AS sentencia_select
[WITH CHECK OPTION [CONSTRAINT restricción]]
[WITH READ ONLY [CONSTRAINT restricción]]
```

- **OR REPLACE:** lo utilizamos si la vista ya existe, la cambia por la actual.
- **FORCE:** aunque no se disponga de los datos necesarios para realizar la consulta, crea la vista.
- **Lista\_columnas:** listado de las columnas que devuelve la consulta.
- **WITH CHECK OPTION:** ofrece la posibilidad de añadir (“INSERT”) o modificar (“UPDATE”) las filas a visualizar.
- **WITH READ ONLY:** vista de solo lectura con posibilidad de asignarle un nombre.

Una vez que ya se han creado las vistas se pueden utilizar de la misma forma que si fueran tablas.

## Veamos un ejemplo

Crear una vista con el nombre y apellidos de los empleados que trabajen en el departamento de ropa de unos grandes almacenes:

```
CREATE OR REPLACE VIEW vista_ejemplo_ropa
AS      SELECT nombre, apellidos
        FROM empleados
        WHERE depot = ropa;
```

Cuando definimos una vista es habitual que usemos la sintaxis “CREATE OR REPLACE”. Por lo tanto se puede utilizar para crear o modificar una vista ya definida previamente.

- **Muestra vista de vistas**

Existe en el diccionario de datos de ORACLE el comando “USER\_VIEWS”, este nos muestra un listado con todas las tablas que posee el usuario actual.

- **Borrado**

Sintaxis para borrar una tabla:

```
DROP VIEW nombre_vista;
```

- **Tipos**

- **Restricciones de clave principal**

La clave principal de una tabla, en la mayoría de los casos, suele ser una columna o una combinación de columnas que van a tener una serie de valores únicos y de esta manera ofrecer integridad.

La restricción de clave principal, cuando intervienen más de una columna, puede que se encuentre con estos valores duplicados en una misma columna así que, para cada combinación de valores, debemos definir esta restricción de clave principal.

La tabla de nuestro ejemplo debe cumplir:

- La restricción de clave principal sólo va incluida en una tabla.
- La clave principal debe ser menor a 16 columnas que es la longitud máxima.
- El índice que genera la restricción de clave principal debe encontrarse entre 1 y 999.
- Las columnas generadas con la restricción de clave principal deben ser asignadas con valores nulos.

- **Restricciones de clave externa**

Hacen referencia a una o varias columnas que van a ser utilizadas para crear diferentes vínculos entre las distintas tablas. La intención es que éstos puedan ser almacenados en una tabla de clave externa. Podemos decir que para que se cree vínculo entre dos tablas, una columna de una de las tablas debe hacer referencia a otra columna que actúa como clave principal de la otra.

Deben cumplir:

- Sólo admite operaciones de “DELETE” cuando tiene más de 253 referencias de clave.
- Cuando una tabla se referencia a sí misma sigue teniendo 253 referencias de clave.
- Más de 253 referencias no hay disponibles.

